UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

Mejorar la experiencia de usuario de delegación de firma digital en la administración electrónica pública

Improving the user experience of digital signature delegation in the public electronic administration

Realizado por
Marco Antonio Hurtado Bandrés

Tutorizado por
Isaac Agudo Ruiz

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, junio de 2021

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

Mejorar la experiencia de usuario de delegación de firma digital en la administración electrónica pública

Improving the user experience of digital signature delegation in the public electronic administration

Realizado por
Marco Antonio Hurtado Bandrés

Tutorizado por
Isaac Agudo Ruiz

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, junio de 2021

Resumen

Actualmente, la Administración Electrónica Pública española está haciendo todo lo posible por actualizarse a las normativas europeas en lo que a identidad digital se refiere, con diversas herramientas que tienen el objetivo de ser más interoperables y sencillas de usar, y así promover la digitalización y agilización de los trámites administrativos. No obstante, a la hora de delegar la identidad digital, estas herramientas no cubren todos los aspectos de seguridad a tener en cuenta ni se suelen tomar medidas para garantizar que quien utiliza la identidad delegada tiene el permiso del delegante en todas y cada una de las ocasiones. Tampoco se conoce que se empleen mecanismos de rastreo de la actividad, por lo que la delegación se convierte muchas veces en un ejercicio de confianza.

Por tanto, los objetivos de este trabajo son, en primer lugar, recalcar la importancia de garantizar que el proceso de delegación de identidad digital (autenticación, firma, etc.) sea lo más seguro posible, en segundo lugar, analizar las novedades de la normativa europea eIDAS y las referencias que hace a la delegación de firma para deducir qué necesitarían las soluciones de delegación de firma para que se adaptaran a ella, y en tercer lugar, analizar la seguridad que otorgan a este proceso las herramientas de identidad digital de la administración pública y de empresas privadas, con las ventajas e inconvenientes que acarrean.

Finalmente, el último objetivo será desarrollar una prueba de concepto de solución de identidad digital con soporte para delegación de firma que aplique los requisitos obtenidos de los análisis previos.

Palabras clave: Delegación de firma, identidad digital, criptografía, Web-eID.

Abstract

Nowadays, the Spanish Electronic Public Administration is doing its best to fit the new digital identity European law by using several tools which have the main purposes of maximizing interoperability and being as simple to use as possible, in order to speed-up and promote the digitization of administrative transactions. Nevertheless, when delegating digital identity, those tools do not provide all the security measures, neither they assure the delegate has the delegator's explicit permission in each usage. In addition, it is not known that those solutions feature any kind of tracking mechanism. Therefore, normally digital identity delegation becomes a trust exercise.

Because of all these reasons, the main aims of this final year dissertation are, first of all, to raise awareness about the importance of protecting the digital identity delegation process (authentication, digital signature, etc.). Secondly, to analyze the new aspects of the European law eIDAS, as well as its mentions to digital signature delegation in order to deduce what a digital signature solution should include to fulfill its requirements. Thirdly, to study how secure is the delegation process with the tools provided by both the Spanish Public Administration and private companies, with their advantages and disadvantages.

Finally, the last purpose is to develop a proof of concept which consists of a digital identity solution with signature delegation support that fits the requirements previously collected.

Keywords: Signature delegation, digital identity, cryptography, WebeID.

Índice

1 Introducción

1.1. Motivación

La principal motivación de este TFG ha sido, en primer lugar, poner de relieve la importancia de garantizar que un proceso tan habitual a día de hoy en la Administración Electrónica Pública como es la delegación de firma, sea el delegado un administrador o una persona de confianza, se lleve a cabo de la forma más segura posible para evitar el uso indebido de la identidad digital de los ciudadanos, lo cual puede acarrear consecuencias graves.

En base a esta premisa, se ha decidido realizar un análisis de algunas de las principales soluciones de identidad digital tanto públicas como privadas y determinar qué grado de seguridad aportan a la delegación de firma, además de analizar las fortalezas, debilidades y mejoras que podrían incorporar para cubrir este proceso de forma más segura.

Finalmente y a partir de dicho análisis, otra motivación ha sido detallar los requisitos de una solución que no solo sea segura, sino también usable y adaptada a la normativa actual y su espíritu de interoperabilidad entre los países de la Unión Europea, para luego desarrollar una prueba de concepto que aplique dicha solución.

1.2. Objetivos

El objetivo principal de este TFG tal y como reza el título, es buscar **formas de mejorar sustancialmente la experiencia de usuario en el proceso de la delegación de firma en la Administración Electrónica Española**, esto es, maneras de volverla más segura, usable, cómoda, etc. Para esta finalidad principal, la división en objetivos es la siguiente:

1. **Investigar la normativa actual** de identidad digital y firma electrónica a nivel europeo (eIDAS) y español (nodo eIDAS español) y analizar las mejoras que aporta respecto a la legislación anterior, así como las menciones a la delegación de identidad digital.

2. **Estudiar algunas de las principales soluciones actuales de identidad digital**, tanto públicas como privadas y analizar el soporte que ofrecen para delegación de firma, exponiendo sus ventajas e inconvenientes y proponiendo mejoras para la finalidad perseguida.

3. En base al estudio de las soluciones, **determinar una serie de requisitos** a cumplir por las distintas herramientas para mejorar la experiencia de delegación de firma.

4. Desarrollar una o varias **pruebas de concepto** que cumplan los requisitos propuestos.

1.3. Estructura del documento

La estructura de este Trabajo de Fin de Grado obedece a la estructura de objetivos expuesta anteriormente: en primer lugar se encuentra la **investigación previa** a la propuesta de requisitos y el desarrollo de la prueba de concepto, y contiene por un lado el estudio de la legislación actual, y por otro lado el mencionado análisis de las soluciones de identidad digital.

A continuación, se expone la **metodología** empleada para el desarrollo de la prueba de concepto, dividida en **fases previas** de captura y análisis de requisitos, diseño y **dos fases de implementación**, documentando todo el desarrollo de la misma. Para concluir, se expondrán las **conclusiones y líneas futuras** de investigación a raíz los avances realizados en este TFG.

1.4. Tecnologías utilizadas

Brevemente se detallarán las tecnologías utilizadas según categorías:

- Lenguajes
    - **Python:** en su versión 3.9.2, para el desarrollo del componente de la aplicación nativa (secciones 6 y 7), debido a que permitía reproducir la funcionalidad de la aplicación en C++ del proyecto Web-eID tomada como base de forma más comprensible, rápida e igualmente funcional.
    - **TypeScript:** para la modificación de la extensión web (secciones 6 y 7).
    - **Java:** para la modificación y el desarrollo de las dos aplicaciones web: el servidor de resgistros y la aplicación web de ejemplo de Web-eID modificada (secciones 6 y 7).

- **HTML, JavaScript y CSS:** en menor medida, para desarrollar y modificar las vistas de las aplicaciones web (secciones 6 y 7).
- **C++:** en un principio se valoró para implementar la modificación de la aplicación nativa, pero finalmente se escogió Python. Por tanto no se ha empleado para el desarrollo de ningún elemento, pero ha sido necesario para la comprensión del funcionamiento de la aplicación nativa original de Web-eID (sección 6).

- Librerías
  - **Cryptography y PyCryptodome:** librerías de Python para operaciones criptográficas (secciones 6 y 7).
  - **Dotenv:** librería de Python para cargar archivos *.env* de variables de entorno (secciones 6 y 7).
  - **PyJwt:** librería de Python para crear tokens JWT (secciones 6 y 7).
  - **Otras librerías de Python:** Aenum (para definir enumeraciones), validators (para validar URLs) y requests (para realizar peticiones web) (secciones 6 y 7).
  - **Web-eid-authoken-validation-java [1]:** librería del proyecto Web-eID para validar tokens JWT (secciones 6 y 7).
  - **Digidoc4j [2]:** librería de seguridad de Java utilizada en este caso para la creación de contenedores ASIC (sección 6.4).
  - **JSON:** librería de Java para trabajar con archivos y formatos Json (sección 7).
  - **JavaMail [3]:** librería de Java para envío de correos electrónicos (sección 7).
  - **nimbus-jose-jwt [4]:** librería de Java para utilizar y validar tokens JWT (sección 7).

- Frameworks
  - **SpringBoot [5]:** para el desarrollo de aplicaciones web en Java (secciones 6 y 7).

- IDEs
  - **Visual Studio [6]** en sus versiones 2015 y 2018, utilizado en la fase de valoración de diseños de la prubea de concepto para compilar los controladores del lector de

tarjetas virtual. No se ha empleado realmente para el desarrollo de la prueba de concepto en sí, sino para el estudio de las alternativas de diseño (sección 5.1).

- **Segger Studio [7]** al igual que visual studio, se empleó en la fase de valoración de diseños como posible IDE de desarrollo de una aplicación nativa que funcionase en el Nordic Thingy:91[1] (sección 5.1).
- **Visual Studio Code [8]** para el desarrollo de la aplicación nativa (secciones 6 y 7).
- **WebStorm [9]:** para todo lo relacionado con la extensión (secciones 6 y 7).
- **IntelliJ [10]:** para todo lo relacionado con lenguaje Java (secciones 6 y 7).

- Otros:

  - **Git y Github:** Git es una herramienta de control de versiones en local, que se puede conectar con los repositorios remotos de Github. El desarrollo de la prueba de concepto ha dado lugar a cuatro repositorios de código, uno por cada componente de la solución.
  - **nRF Connect Suite [11]**. Esta suite se valoró e instaló para estudiar su viabilidad en términos de tiempo y complejidad en el desarrollo de la prueba de concepto (sección 5.1).
  - **MongoDB Atlas [12]:** cloud DBaaS (DataBase as a Service) de Mongo para poder acceder a la base de datos del servidor de registros de forma remota. Desde SpringBoot se ha enlazado la aplicación web con la base de datos remota (sección 7).
  - **MongoDB Compass [13]:** aplicación de escritorio de MongoDB para gestionar las bases de datos mediante una interfaz de usuario (sección 7).
  - **Cliente @firma [14]:** se ha utilizado el proyecto de código abierto del cliente de @firma para su estudio y para hacer pruebas de integración con el proyecto de Web-eID (sección 6).

[1] https://www.nordicsemi.com/Software-and-tools/Prototyping-platforms/Nordic-Thingy-91

2

Investigación previa

Esta parte del TFG se centra principalmente en el estudio del estado legal de la identidad digital más relacionado con la firma y la delegación de firma a nivel europeo y nacional, y de las herramientas ofrecidas por entidades públicas y privadas (principalmente prestadores de servicios de confianza) a nivel nacional. El objetivo principal de este estudio es determinar una serie de requisitos y proponer mejoras de lo existente para hacer más seguro el proceso de delegación de firma.

2.1. Legislación

El reglamento actual principal es eIDAS (electronic IDentification, Authentication and trust Services) a nivel europeo, que tiene como principales objetivo establecer un **marco legal común** para todo lo relacionado con identidad digital (firmas electrónicas, sellado de tiempo, autenticación, etc.), así como una serie de pautas que los prestadores de servicios de confianza deben cumplir en todos los estados miembros [15].

Aquí ya aparece una novedad, pues el término ***servicios de confianza*** hace referencia no solo a la firma electrónica, sino que a partir de este reglamento incluye también al sellado de tiempo, los documentos electrónicos, los servicios de entrega electrónica certificada, y los servicios de certificados para la autenticación de sitios web [15]. A raíz de estos términos aparece otro relacionado que es el de ***prestadores de servicios de confianza*** y se refiere a aquellas entidades que ofrecen estos servicios.

En cuanto a su naturaleza jurídica, al no tratarse de una directiva, se aplica directamente, reduciendo la posible heterogeneidad de aplicación e interpretación por parte de los estados miembros. Esto tiene sentido, puesto que como se ha mencionado, se pretende eliminar las

barreras ya sean tecnológicas o administrativas en los trámites que hacen uso de la identidad digital entre los estados miembros, fomentando la interoperabilidad y agilizando los procesos tanto a nivel nacional como entre países, haciéndolos más eficientes. Un ejemplo es la intención de poder usar el Documento de Identificación Electrónica en todos los estados miembros [15].

Respecto a las **novedades** que aporta el reglamento eIDAS, en primer lugar, mantiene los tres tipos de firma ya contemplados: simple, avanzada y avanzada basada en un certificado reconocido, aunque renombrando esta última a *firma electrónica cualificada*. La novedad en este caso está en que la firma electrónica cualificada, al ser de los tres tipos de firma la más segura, debe ser **reconocida en todos los estados miembros**[15]. Aún así, la firma electrónica avanzada puede ser la más útil en la mayoría de ocasiones al ser por un lado más segura que la simple, y por otro lado no requerir un dispositivo cualificado de creación de firmas. Los principales **efectos jurídicos** de esta novedad son principalmente[15]:

- El **reconocimiento de la validez jurídica** de la firma electrónica incluso si es simple.
- La **equivalencia** entre la firma electrónica (incluso si es simple) y la firma manuscrita.
- El **reconocimiento de la firma electrónica cualificada** en todos los estados miembros de la Unión Europea.

Continuando con los **contenidos y novedades** del reglamento, los más significativos son [16]:

- **Reconocimiento mutuo (Artículo 6):** Describe las **condiciones** en las cuales se reconocerá el medio de identificación electrónica en otro Estado miembro, siendo por ejemplo una de ellas que el medio de haya sido expedido por un sistema de identificación electrónica incluido en la lista publicada por la Comisión de conformidad con el artículo 9. El artículo 9 profundiza, entre otros aspectos, en los **contenidos** que deben transmitir los Estados miembros a la Comisión, que incluye la descripción del sistema de identificación, las autoridades responsables de él, las entidades que gestionan el registro de los datos únicos de identificación de los ciudadanos, etc.
- **Cooperación e interoperabilidad (Artículo 12):** Define elestablecimiento de un marco de interoperabilidad que debe ser **tecnológicamente neutro**, esto es, sin discrimi-

nación entre soluciones técnicas, ajustado a normas internacionales y europeas, que facilite la aplicación del principio de privacidad desde el diseño[2] y que se ajuste a las normativas europeas de de protección de datos.

Sin embargo, aunque se haga hincapié en la neutralidad tecnológica, en este artículo se describen unos **requisitos técnicos mínimos** necesarios a nivel de seguridad y de interoperabilidad para los prestadores de servicios de confianza, y especialmente para los que ofrecen servicios cualificados.

- **Listas de confianza (Artículo 22):** por la mención en el punto siguiente, es conveniente señalar que este artículo determina, entre otros aspectos, que los Estados miembros deben mantener y actualizar listas de confianza, que incluyen información sobre los prestadores cualificados de servicios de confianza y sobre los propios servicios prestados.

- **Etiqueta de confianza «UE» para servicios de confianza cualificados (Artículo 23):** se trata de un **reconocimiento** otorgado a los prestadores de servicios de confianza del Artículo 22 para indicar los servicios que ofrecen, garantizando el acceso a la lista de prestadores cuando se muestre esta etiqueta. También define aspectos como la presentación, composición, tamaño y diseño de la misma. Aunque parezca un artículo de menor importancia, es de gran utilidad para dar confianza a aquellas personas con menos experiencia en el ámbito tecnológico y de seguridad que accedan a servicios de confianza por primera vez.

- **Efectos jurídicos de las firmas electrónicas (Artículo 25):** afirma los aspectos principales mencionados sobre la firma electrónica a nivel jurídico: la **validez idéntica** incluso como prueba en procedimientos judiciales y aunque no sea cualificada con respecto a la firma manuscrita y el **reconocimiento** de la firma electrónica cualificada en todos los estados miembros.

- **Requisitos para firmas electrónicas avanzadas (Artículo 26)** descritos anteriormente.

- **Sellos electrónicos (Artículos de la sección 5):** Desde el primer artículo de esta

[2] Explicación de *privacidad desde el diseño*

sección (artículo 35) se definen los **efectos jurídicos** de los sellos electrónicos. Estos incluyen, de forma similar al caso de la firma electrónica cualificada y en el caso del sello electrónico cualificado, la presunción de **integridad y no repudio origen** de los datos a los que va asociado, así como la **validez** de estos entre estados miembros.

- **Servicio de entrega electrónica certificada (Artículos de la sección 7):** Desde el artículo 43 se garantiza su admisión en procedimientos judiciales, y en el caso de que sean cualificados se **presupone la integridad de los datos** al contar con elementos como un remitente identificado y un certificado del proceso de entrega.

- **Autenticación de sitios web (Artículo 45 y Anexo IV):** Desde el primer artículo de esta sección (artículo 45) se establecen, entre otros aspectos, **requisitos** que deben satisfacer los certificados cualificados de autenticación de sitios web, como pueden ser desde indicar en un formato adecuado cuándo fue expedido en calidad de cualificado, hasta el lugar donde están los servicios para consultar su estado.

Por tanto, se puede resumir las principales aportaciones novedosas del reglamento eIDAS en:

- **El establecimiento de condiciones para el reconocimiento de los medios de identificación** electrónica de personas físicas y jurídicas en los estados miembros de la Unión Europea.

- **La obligación de contar con regímenes de supervisión y responsabilidad** con el objetivo de asegurar los criterios cualitativos de los servicios ofrecidos por los prestadores de servicios de confianza cualificados a nivel europeo.

- **Proporcionar un marco legal para todos los servicios de seguridad**, que ahora también incluyen otros como el sellado de tiempo electrónico y los servicios de entrega certificada.

- **Obligar a los Estados miembros a garantizar la posibilidad de acceso con los medios de identificación electrónica (eIDs)** a todos los ciudadanos de los Estados de la Unión Europea.

Las consecuencias son claramente positivas para otros objetivos que se venían persiguiendo anteriormente, como puede ser el de establecer un **mercado único digital** o **favorecer el eGovernment** [17] al simplificar los trámites y aumentar la interoperabilidad, la movilidad y la transparencia, mejorando significativamente la experiencia de usuario.

Aunque eIDAS sea de 2014, Europa sigue a día de hoy trabajando en los objetivos iniciados y potenciados por este Reglamento. Un gran ejemplo de esto se materializa en la *Brújula Digital* de la UE para 2030, que sirviéndose de eIDAS como base, propone una serie de objetivos entre los que figuran, por ejemplo, que todos los servicios públicos importantes estén disponibles en línea o, que para esas fechas, el 80 % de los ciudadanos utilicen soluciones de identidad digital [18].

Una proposición reciente (junio de 2021) que ayudará a conseguir estas metas consiste en un **marco de trabajo para la Identidad Digital Europea.** Este marco de trabajo permitirá a todos los ciudadanos europeos utilizar sus carteras digitales de la misma forma en todos los países para realizar diversos trámites y acceder a servicios, desde universitarios hasta aquellos para adquirir viviendas. Las carteras serán proporcionadas por entidades públicas o privadas siempre que sean reconocidas por un Estado miembro [18]. Esta Identidad Digital Europea debe estar **disponible para cualquier persona física o jurídica** que desee utilizarlo, debe ser **reconocida en toda la Unión Europea** y debe otorgar al usuario un gran **control sobre sus datos** [18]. Para conseguir esto, en el corto plazo se pondrá en marcha una *caja de herramientas* elaborada por los Estados miembros con aspectos más técnicos como la arquitectura y los estándares.

2.1.1. Nodo eIDAS español

Los organismos responsables del Nodo eIDAS español son el Ministerio de Asuntos Económicos y Transformación Digital, la Secretaría de Estado de Digitalización e Inteligencia Artificial y la Secretaría General de Administración Digital[19]. La principal finalidad de sus actividades es adaptar el reglamento eIDAS en España, lo que se traduce en:

- **Aceptar los medios de identificación electrónicos de otros países en la administración española**, así como hacer que los medios españoles cumplan los requisitos establecidos en el reglamento para que sean aceptados en el resto de Estado miembros.

- **Interoperabilidad**: independientemente de la aceptación de los sistemas de identificación, estos deben poder comunicarse sin problemas. Y esta comunicación debe ser posible tanto entre los sistemas nacionales (distintas administraciones cuentan con distintos sistemas) como entre los sistemas internacionales.

Para ambos fines, se ha adaptado la implementación de referencia de los requisitos técnicos elaborados por la Comisión Europea. El resultado de esta adaptación es todo el sistema Cl@ve [19], el cual se analizará en puntos posteriores de este trabajo.

Respecto al apartado de prestadores de servicios de confianza y obedeciendo al artículo 22 del reglamento eIDAS, el Ministerio de Asuntos Económicos y Transformación Digital ha elaborado una amplia lista con información relativa a los prestadores[3], tanto de servicios cualificados como no cualificados, accesible públicamente.

Todas estas adaptaciones de las intenciones y novedades que aporta el reglamento eIDAS se pueden ver reflejadas en varios **Boletines Oficiales del Estado**. En uno reciente (BOE-A-2021-5032 [20]), se modifican varios artículos y disposiciones del Real Decreto 4/2010 del 8 de enero[4] y que regulaba el Esquema Nacional de Interoperabilidad en el ámbito de la Administración Electrónica con el claro objetivo de fomentarla entre sistemas nacionales:

- El artículo 16, referido a **condiciones de licenciamiento aplicables** menciona que para las aplicaciones desarrolladas por la Administración pública, "el fin perseguido es el aprovechamiento y la reutilización de recursos públicos" [20], pudiendo adaptarse o ampliarse para ser usado en otra administración. También hay referencias al código abierto, que puede ser una buena iniciativa para favorecer la estandarización y comunicación entre los sistemas de las administraciones nacionales, pues unas pueden analizar el funcionamiento de las aplicaciones y herramientas desarrolladas por otras, e incluso adaptarlas y mejorarlas, resultando en nuevas versiones que continuarán siendo de código abierto.

- El artículo 17 directamente menciona los **directorios de aplicaciones de libre reutilización**, que podrán usar todas las Administraciones Públicas ya que deben estar

[3] Lista de prestadores de servicios de confianza

[4] Real Decreto 4/2010, de 8 de enero

conectados. También profundiza en el contenido de los directorios.

- El artículo 18 habla específicamente de la **interoperabilidad en la política de firma electrónica y de certificados**, que se alcanza mediante la definición de un "marco general de interoperabilidad para el reconocimiento mutuo de las firmas electrónicas basadas en certificados de documentos administrativos en las Administraciones Públicas" [20].

- La Disposición adicional primera referida al **Desarrollo del Esquema Nacional de Interoperabilidad** menciona, en primer lugar, una serie de **normas técnicas** de interoperabilidad que deben cumplir las Administraciones Públicas (estándares, metadatos, formatos...). En segundo lugar, se explican los **instrumentos** a desarrollar para ganarantizar la interoperabilidad, que incluyen el Sistema de Información Administrativa (que hace de inventario de información relativa a servicios, procedimientos y otras actuaciones), el Centro de interoperabilidad semántica de la Administración (que tiene la finalidad de "facilitar la comprensión semántica de los servicios de intercambio de datos de las Administraciones y maximizar la reutilización de activos semánticos en la construcción de éstos" [20]), el Centro de Transferencia de Tecnología (que es el mencionado Directorio de aplicaciones de libre reutilización) y el Directorio Común de Unidades Orgánicas y Oficinas de las Administraciones Públicas (que sincroniza los sistemas que tratan la información previa).

  Además menciona aspectos legales a la hora de actualizar las normas técnicas y los organismos competentes, como es por ejemplo el Centro Criptológico Nacional.

2.2. Delegación de firma

La delegación de firma se puede definir de forma sencilla como ***dar la facultad a otra persona para que firme en tu nombre*** [21].

En el Artículo 12 del BOE del 2 de Octubre de 2015 (Ley 40/2015, de 1 de octubre, de Régimen Jurídico del Sector Público [22]) se utiliza este concepto en el ámbito de los órganos administrativos, señalando que estos podrán delegar la firma "en los titulares de los órganos o unidades administrativas que de ellos dependan" [22]. También se señala que el hecho de que la firma sea delegada, por un lado, se debe hacer saber, y por otro lado, que es tan válida

como una firma original [22]. Cabe señalar que no son equivalentes los términos *delegación de firma* y *delegación de competencias*, pues el primero es una consecuencia y está incluido en el segundo, es decir, que delegar la firma es una de las competencias que se puede delegar[23]. La aplicación del reglamento eIDAS en este aspecto, por tanto, tendrá el efecto de que **una firma electrónica delegada deberá tener la misma validez que una firma delegada manuscrita**.

Las principales **ventajas** del acto de delegación de firma se puede considerar que son:

- **Ahorro de tiempo y agilización de trámites.** Tanto en entidades mayores ya sean empresas u órganos públicos, donde los ocupantes de cargos superiores delegan la firma en otros cargos, como en el día a día de los ciudadanos, que pueden delegar su firma en un administrador para que realice los trámites por ella, la delegación de firma permite descongestionar el tráfico de transacciones administrativas.

- En el ámbito empresarial, la delegación de firma **otorga una responsabilidad al delegado**, pudiendo resultar en un incremento de motivación de este a la hora de hacer su trabajo.

- **Movilidad.** Tanto en el ámbito empresarial y de administraciones como en el cotidiano, delegar la firma permite poder **firmar desde cualquier sitio**, tanto si el delegado cuenta con todo lo necesario para ello como si no, pues por los medios telemáticos seguros que ofrecen los prestadores de servicios de confianza, el delegante puede proporcionarlos sin miedo alguno.

- **Organización de la documentación y ahorro de papel**, aunque sea más consecuencia directa de la transformación digital promovida por la propia delegación de identidad digital, la delegación de firma contribuye en gran medida a la reducción del impacto ecológico de los documentos físicos asociados a trámites administrativos.

En cuanto a los **inconvenientes**, encontramos los siguientes:

- **Dependencia de la confianza en el delegado.** Con los medios y herramientas actuales, tal y como se analizará en el siguiente apartado, cuando se delega la firma electrónica,

se está delegando la identidad digital, y a pesar de que existan contratos de por medio, condiciones y acuerdos entre el delegante y el delegado y algunas medidas de seguridad, no suelen ser suficientes, dándose por hecho que toda firma delegada en todas y cada una de las ocasiones se ha hecho con el consentimiento del la persona original.

- **Dependencia de las medidas de seguridad particulares.** Como consecuencia de lo comentado en el punto anterior, y suponiendo el caso de contar con medidas de seguridad que garanticen aspectos como el consentimiento explícito en cada ocasión o el rastreo de actividad, se depende de unas medidas de seguridad centralizadas y de la confianza en el proveedor de estos, es decir, en que toda la actividad de éste tenga fines lícitos y en que el sistema sea realmente robusto para soportar todo tipo de ataques, pues la información gestionada es extremadamente sensible.

2.3. Análisis de las soluciones de empresas privadas

Las principales soluciones que se analizarán son las basadas en la nube (*cloud*), pues son las que ofrecen mayor soporte y están en principio más pensadas para proteger la delegación de identidad digital. No obstante, antes conviene nombrar otros dos tipos de soluciones que utilizan otras arquitecturas:

- **Extensiones:** las soluciones de identidad digital basadas en extensiones son la evolución de las applets, no soportadas por las versiones nuevas de los navegadores a día de hoy. Su funcionamiento consiste principalmente en la **comunicación con aplicaciones nativas** (de escritorio) **mediante paso de mensajes nativo** [24]. Las aplicaciones nativas acceden, por ejemplo, al almacén de certificados del ordenador y se hacen las operaciones de firma mediante JavaScript en el navegador. Estas soluciones son útiles también para el caso de uso de utilizar un token USB a modo de almacén de certificados pero que no tenga capacidad de firmar, o en el caso de contar con una tarjeta inteligente (*Smartcard*) o token hardware que pueda realizar operaciones criptográficas, pero en este último escenario será el token hardware el encargado de realizar las operaciones. Por tanto, **la solución no está compuesta simplemente por una extensión**, sino que se necesitan más componentes software, como una aplicación nativa y una aplicación web compatible con la extensión.

Estas soluciones hacen posible la **delegación de firma en la web** ya que el delegado puede utilizar los certificados, ya estén en el almacén de certificados del sistema operativo directamente o de un token hardware para hacer trámites en internet en nombre del delegante. Además, en este último caso no es necesario instalar los certificados en el ordenador ni en el navegador. Algunos ejemplos de este tipo de solución son Signer.Digital [25] y Web-eID [26].

- **Soluciones hardware:** las soluciones de identidad digital centradas en hardware más extendidas son los **tokens hardware**, que incluyen las **tarjetas inteligentes** (*smartcards*), pues sin ir más lejos, a día de hoy los documentos de identificación electrónicos, incluidos los europeos (eIDs) suelen ser smartcards. Estos dispositivos cuentan con procesadores para realizar operaciones criptográficas como la firma electrónica, y la delegación de firma en este caso es tan sencilla como proporcionar el token hardware al delegado, eso sí, junto con una contraseña (PIN normalmente) que protege las claves almacenadas. Las smartcards por su parte necesitan lectores compatibles y que estos tengan drivers soportados por el sistema operativo pertinente, aunque a día de hoy existen soluciones que permiten sustituir los lectores de tarjetas por los chips NFC de los smartphones.

  No obstante, existen **otras soluciones hardware** como pueden ser los OTP (One-Time Password) Tokens, que generan códigos de un solo uso, o incluso memorias USB protegidas de forma biométrica (huella dactilar por ejemplo).

  Estas últimas soluciones permiten un **mayor control**, sobre todo si en el caso de los tokens con seguridad biométrica se pueden añadir varias huellas, pero son soluciones **más complejas** de implementar a nivel tecnológico y de integrar con una infraestructura. Algunos ejemplos de cada tipo de token son los de Microcosm [27].

Por tanto, aunque todas las soluciones propuestas hasta ahora permiten realizar la delegación de firma, **las que ofrecen un mayor control sobre la firma delegada son las soluciones hardware**. No obstante, las soluciones basadas en extensiones podrían controlarlo, por ejemplo, manteniendo un registro del uso de los certificados, con *blacklists* o *whitelists* de IPs, etc. Se profundizará en estas modificaciones en el apartado de mejoras de las soluciones y del diseño de la solución de la prueba de concepto.

Comenzando con las soluciones basadas en la **nube**, todas tienen elementos de funcionamiento comunes, como **almacenar el certificado en un entorno cloud seguro**, controlando y registrando el acceso a los certificados. Las soluciones que ilustran esta arquitectura son:

- **FirmaCloud (Grupo 2000):** se basa en que el certificado, una vez emitido, se almacena en la nube directamente, permitiendo utilizarlo desde cualquier dispositivo y en cualquier lugar, y **delegar su uso a terceros**. Para controlar la delegación, ofrece las funcionalidades de **registro de uso**, quedando constancia de la actividad tanto si el actor es el delegante como el delegado, y de **limitación de uso** del mismo por parte de terceros.

  FirmaCloud también ofrece otros servicios extra como avisos de expiración, revocación *one-click* o la integración en aplicaciones vía api [28].

  Grupo 2000 también cuenta con una **solución de firma online desde móviles** (Firmafy), que funciona de forma que delegado envía un enlace de Firmafy vía email, WhatsApp o SMS a quien debe firmar el documento, que recibirá un OTP para poder realizar la firma. Este funcionamiento también permite varios firmantes [29].

- **ViaFirma Inbox (Viafirma):** Viafirma Inbox es un portafirmas para organizar firmas delegadas y establecer permisos en los niveles de:

  - Revisión: el delegado puede, por ejemplo, ver tareas pendientes establecidas por el delegante, ordenar, comentar…
  - Visto bueno: permiso de revisión al que se le añade poder dar visto bueno a tareas.
  - Firma: todos los permisos anteriores, al que se le suma el poder usar la firma electrónica delegada.

  Esta organización en niveles de privilegios se puede hacer con la firma previa de un documento de acuerdo entre ambos usuarios, pudiendo incluirse este cada vez que se utiliza la firma delegada, ratificando que existe la autorización y el consentimiento explícito de uso de la firma.

  Otras características incluyen el soporte de los tres tipos de firma digital (simple, avanzada y cualificada), definición de flujos de firma visuales, sellado de tiempo con fecha y hora de la firma, y los metadatos tanto fijos (gestionados por administradores) o libres

añadidos al crear la solicitud de firma. Además, Viafirma ofrece una una aplicación para smartphones que permite firmar desde móvil y soporta la solución de firma en la nube de Viafirma, que se explicará a continuación [30].

- **Viafirma Fortress (Viafirma):** con un funcionamiento similar al de FirmaCloud, almacena los certificados en un servidor seguro para hacerlos disponibles en cualquier lugar. Para acceder a estos certificados, el usuario se puede autenticar por medios como contraseña, tarjeta de claves, locución o factores biométricos, entre otros. También cuenta con una aplicación de escritorio [31].

- **Docuten:** se trata de una *solución* distinta, pues su funcionamiento se basa en que el usuario delegue la firma en la plataforma, para que esta firme por él haciendo uso de una Autoridad de Certificación de confianza. Por tanto, el delegado es la propia solución. Asimismo, se señala el ahorro de tiempo, pues el documento se firma automáticamente cuando se envía [32].

En definitiva, las soluciones cloud permiten, por ejemplo, **delegar firma a trabajadores que hacen gestiones para una empresa o administración pública**, **controlar que las gestiones cumplan con la legalidad vigente** y **almacenar el certificado en un sitio centralizado seguro.**

2.4. Análisis de las soluciones de la Administración Electrónica Pública

A continuación se estudian algunas de las principales soluciones de identidad digital que utilizan día a día tanto ciudadanos como administraciones públicas. Una lista más extensa y detallada de los mismos se puede consultar en el catálogo de soluciones ofrecido por el Portal de la Administración electrónica, en la sección de *identidad digital y firma electrónica*[5]

[5] https://administracionelectronica.gob.es/pae_Home/pae_Estrategias/Racionaliza_y_Comparte/catalogo-servicios-admon-digital.html

2.4.1. Sistema Cl@ve

La infraestructura clave cuenta tanto con soluciones para autenticación como para firma, y los componentes que la forman están disponibles por distintos medios, desde web hasta smartphone. Uno de los objetivos que persigue es **establecer un sistema común de identidad digital**, evitando que cada Administración deba implementar y gestionar la suya propia [33], lo cual afecta en consecuencia positivamente a los ciudadanos, aumentando la comodidad al realizar los trámites administrativos.

Gracias a los distintos tipos de Cl@ve (analizados a continuación), se pueden establecer **niveles de seguridad**, permitiendo la realización de determinados trámites solo a los autorizados, o de la misma forma permitiendo al ciudadano elegir el tipo de autenticación en función del trámite que desea realizar. También ofrece soporte para el DNIe.

Para la **autenticación**, Cl@ve ofrece las siguientes alternativas [34]:

- **Certificados digitales y Cl@ve Certificado electrónico**, siempre que estén soportados por la suite @firma y las plataformas de validación, es decir, son aquellos obtenidos de los proveedores de servicios de confianza. Esto también deja ver el espíritu de interoperabilidad entre sistemas nuevos y existentes, de forma que se puedan ir actualizando herramientas sin que las anteriores queden inmediatamente obsoletas.

- **DNI electrónico / Cl@ve DNIe**: requiere algunas de las aplicaciones mencionadas en el apartado de *otras herramientas* más adelante para lectura y los módulos criptográfico necesarios en los navegadores.

- **Cl@ve PIN**: PIN de uso único que funciona como OTP, obtenido a través de la aplicación Cl@ve PIN y cuyo acceso puede llevarse a cabo mediante factores biométricos.

- **Cl@ve permanente**: para personas que acceden normalmente a servicios electrónicos de la administración pública. Consiste en una combinación de un código de usuario, el número del DNI y una contraseña, a la que se añade un OTP enviado por SMS en los casos de mayor seguridad. En este servicio deberá estar registrado el ciudadano para realizar la firma centralizada [35], para que su certificado de firma quede asociado al documento físico.

- **Cl@ve Ciudadanos UE**: este método satisface en parte los objetivos de interoperabilidad y compatibilidad de la UE, pues un ciudadano con un documento de identificación electrónico europeo (eID) podrá hacer uso de los sistemas de la administración española, al igual que el DNIe será aceptado en los sistemas de otros países.

Para todo lo que tiene que ver con la **firma electrónica** se hace uso de **Cl@ve Firma**. Previamente, el ciudadano debe registrarse en el sistema Cl@ve de autenticación y se deben de haber generado tanto contraseñas en este sistema como claves de autenticación y firma (ya sea el en momento de firmar o en otro momento previo) [36]. La intención del gobierno es ir integrando todas las administraciones poco a poco en este sistema, pues ahora mismo, por ejemplo, hay cierta heterogeneidad en las pantallas mostradas durante el proceso de firma porque cada una custodia un tipo de información determinada (GISS, Agencia tributaria, Dirección General Policía...).

A continuación se analiza el proceso de firma mencionando las etapas del proceso en Cl@ve Firma [36]:

1. **Autenticación** al acceder a servicios que integran el sistema Cl@ve en alguna de sus formas.

2. (Opcional) Actualizar la contraseña.

3. **Identificación**: aunque se haya hecho una autenticación previa, se debe introducir el DNI y contraseña para quedar identificado en sistema cl@ve.

4. **Acceso a trámite**: la disponibilidad de estos dependerá de si se está utilizando Cl@ve PIN o permanente, pues si se trata de un trámite de alta seguridad, se requiere Cl@ve permanente y se enviará un OTP vía SMS.

5. Completar el trámite. Incluye marcar en una casilla que los datos aportados son ciertos.

6. Generación de claves de firma si no se han solicitado dichos certificados previamente.

7. **Proceso de firma**: Introducir la contraseña y el OTP.

8. **Finalización del proceso de firma**: opcionalmente, el usuario puede solicitar y descargar un justificante.

9. Finalización del trámite.

Por tanto, se aprecia que es un proceso con **buenas medidas de seguridad**, requiriendo autenticación, introducción de credenciales e incluso OTPs en los casos más delicados. Además, es flexible al poderse generar las claves de firma en el momento si no existían. No obstante, aunque permite generar un justificante como comprobación del proceso, no se menciona ninguna creación de un registro de actividad que pueda consultar el usuario posteriormente ni al firmar ni al autenticarse, y no en todos los casos es necesario el OTP, por lo que **la delegación de firma queda desprotegida en los trámites *menores***. Las mejoras para este sistema se propondrán en el apartado de mejoras.

Finalmente, añadir que aunque se haya mencionado que los sistemas de firma electrónica, al igual que los de autenticación, admiten los certificados admitidos por @firma y emitidos por prestadores de servicios de confianza, actualmente sólo determinados trámites de la sede electrónica permiten el uso de AutoFirma.

2.4.2. Suite @firma

La solución @firma compone de una serie de productos centrados en la validación y realización de firma con el fin de facilitar su uso en la administración pública. A continuación se mencionan los componentes, analizando el funcionamiento de los más relevantes que han podido ser probados, pues no todos son accesibles por cualquier ciudadano [37] [38] [33]:

- **@firma:** es la solución tecnológica en sí, es decir, **la plataforma de validación de certificados y firmas**. Por tanto, nos encontramos ante una solución más centrada para ser usada por las administraciones en lugar que por los ciudadanos, aunque en seguida se verán formas de uso por parte de los ciudadanos. Es una solución basada en software libre, estándares abiertos bajo licencias GNU GPL versión 2 y EUPL v1.1, y en Java [38], con muchos proyectos disponibles en GitHub[6]. Cuenta con un modelo federado (@firma federada).

[6] Centro de Transferencia de Tecnología - Forja CTT en GitHub

- **TS@:** es la Autoridad de sellado de tiempo del Ministerio de la Secretaria General de Administración Digital.

- **Productos:**

  - **eVisor:** aplicación web para generar copias auténticas e informes de firma en papel desde formato PDF.

  - **Integr@:** conjunto de librerías para favorecer, como su propio nombre indica, la integración con @firma. Algunos módulos, como el de cifrado (*Integra-encryption*) solamente han actualizado la versión desde que se subió el proyecto hace cuatro años[7].

  - **Port@firmas:** se trata de una serie de componentes también para fomentar el uso de la firma electrónica en los flujo de trabajo de las organizaciones, disponible en forma de aplicación web y como aplicación para dispositivos móviles.

  - **FIRe**: se trata de una solución integral de firma, para escritorio, aplicaciones móviles y en la nube. Aunque FIRe es de fuentes abiertas, hay componentes propietarios que deben ser solicitados. Si se quiere descargar e instalar mediante el kit de integración en lugar de compilar el código fuente, es necesario conectarse por Redsara y autenticarse en el portal [39][40].

- **Portales:** componen los denominados *servicios directos al ciudadano*, y son:

  - **Valide:** se trata de un portal web para comprobar servicios web de @firma, centrado principalmente en la validación de firmas y certificados, pudiendo generar justificantes y ver el detalle de la validación. Cuenta con otras funcionalidades como precisamente realizar firmas o validar sedes electrónicas. Sin embargo, muchos de los enlaces y botones de esta herramienta abren la aplicación de escritorio de Autofirma y falla frecuentemente, bien puede deberse a fallos del autoscript o a que no se haya realizado la actualización del miniapplet al autoscript en dicho portal.

  - **Portal de firma**: el objetivo de este portal es principalmente pedagógico, pues sirve para que el ciudadano adquiera conceptos básicos sobre la firma electrónica y su

[7] https://github.com/ctt-gob-es/integra/commits/master/Integra-parent/Integra-encryption

proceso. El objetivo que persigue es positivo, y la interfaz es amigable, con procesos por pasos sencillos de seguir. Algunos problemas que presenta es, por ejemplo, que para enseñar a hacer la firma hace uso de Valide, que como se ha comentado falla con frecuencia. Más allá de eso, es muy positivo que distinga entre usuarios, disponiendo de un apartado para ciudadanos otro para empleados públicos, y otro para empresas.

- **Clientes:** los clientes de la suite vienen en distintas formas en función de los dispositivos y entornos de uso:

  - **Autofirma:** es el **cliente de escritorio**, desarrollado en **Java**, y permite hacer las funcionalidades de firma descritas en Valide: firma de documentos y validación. En el caso de la firma de PDFs, que suele ser muy común, la aplicación permite ubicar la firma en un sitio determinado del documento, así como personalizar la apariencia de la firma, como por ejemplo cambiando el tamaño y la fuente de texto, añadiendo una imagen etc. Todo esto mediante una interfaz de usuario sencilla pero intuitiva. Además, el proceso de verificación de firmas también es sencillo, en pocos pasos y ofreciendo información detallada tanto del certificado con el que se ha realizado la firma, como de los datos firmados.

    Sin duda, se trata de un componente que **se podría conservar para utilizar en infraestructuras de soluciones con soporte seguro para delegación de firma** como las de extensiones, o incluso extenderse para que ofreciera soporte para utilizar tarjetas inteligentes. El código de la aplicación se puede consultar en GitHub y ha sido analizado para una posible integración en la prueba de concepto. A nivel de documentación del código, cuenta con diversas **pruebas** para cada una de las funcionalidades, lo cual ha sido de utilidad a la hora de comprender el funcionamiento de algunos componentes analizados durante el desarrollo de la prueba de concepto.

  - **Miniapplet + Autofirma:** es la solución que permite ejecutar Autofirma desde un navegador. El miniapplet está desarrollado en JavaScript y lanza la aplicación de escritorio. Ahora se ha sustituido por el llamado AutoScript[8]. No obstante y a

[8] https://administracionelectronica.gob.es/ctt/clienteafirma/descargas

raíz de lo probado en Valide, no parece que funcione especialmente bien, pues no se consigue lanzar la aplicación de autofirma con éxito en la mayoría de ocasiones. Como se ha comentado, podría ser que no se haya actualizado al autoscript y continúe con el miniapplet.

- **Firm@Movil:** se trata de una aplicación de smartphones para realizar la firma con dispositivos móviles, algo así como un autofirma para móviles. A raíz de las reseñas y la valoración en general de la aplicación en las distintas tiendas de aplicaciones, se puede inferir que no suele funcionar correctamente.

Como anotaciones finales sobre esta suite, señalar por una parte que los componentes de @firma y TS@ oficialmente están incluidos en una sección de *servicios* de la suite [38]. En dicha sección también se incluye el sistema Cl@ve, por lo que realmente esta suite no funciona de forma independiente, sino que está también pensada para ser utilizada en conjunción con otras soluciones más actuales de la Administración Electrónica Pública, evitando la obsolescencia inmediata. Por otra parte, **la documentación de esta suite puede llevar a confusión en ocasiones**, al hacerse muchas referencias a componentes de la misma que ya están obsoletos, como ocurre con el caso del Miniapplet.

2.4.3. Otras herramientas

- **AutenticA:** enfocada a la autenticación, permite el Single Sign On[9] y la autorización de empleados públicos, personal, etc. en el ámbito de las aplicaciones de las distintas administraciones [41]. Se trata por tanto de un componente no enfocado al uso cotidiano por parte de ciudadanos que no trabajen en las administraciones.

- **Aplicaciones de escritorio del DNIe**, en el portal del DNI electrónico se hace referencia a una aplicación de escritorio para realizar operaciones como firmar con y sin documento adjunto y verificar firmas con y sin documento adjunto tanto online como offline, con soporte de firma múltiple y de acuerdo al estándar CMS con el DNIe [42]. La aplicación se basa por un lado en módulos PKCS#11 de la FNMT-RCM, el software desarrollado en Java para las operaciones criptográficas y los OCSPs de la FNMT-RCM, requiriendo tener el JRE 1.5 instalado.

[9] Definición y funcionamiento del Single Sing On (SSO)

El **flujo de uso de la aplicación** es sencillo: primero se escoge la operación a realizar, luego los documentos a validar o firmar, y tras la introducción del PIN y la confirmación del consentimiento para realizar la operación, se realiza la operación. **Los requisitos de instalación son quizás demasiado técnicos** para mucha gente poco familiarizada con conceptos informáticos, que es gran parte (por no decir la mayor parte) de la población. No obstante, se trata de una aplicación **antigua (2006)**, aunque sigue apareciendo en el portal del DNI electrónico, por lo que se entiende que sigue siendo funcional [43], pero que probablemente no se aconseja su uso. La aplicación cumple su cometido sin funcionalidades adicionales y con interfaces propias de la época en la que se desarrolló.

- **Aplicaciones para smartphones del DNIe**, por un lado, existe una aplicación cuyo fin es permitir el uso del **smartphone como lector NFC**, haciendo el uso del DNIe en el ordenador más sencillo[10]. Cuenta con un asistente que ayuda a que el proceso se complete sin problemas y puede ser útil en varios casos de uso. También existe una aplicación de **firma de PDFs con el DNIe**, que es realmente una mezcla de la aplicación de escritorio anterior y Autofirma, pues permite personalizar la apariencia de la firma dentro del propio PDF. Esta aplicación cuenta, al igual que la anterior, con un asistente que hace el proceso más cómodo [11].

  Finalmente, también existe una aplicación a modo de *hub* de acceso a algunos portales de la administración pública, como la DGT o la Agencia Tributaria, y a portales de servicios locales (actualmente solamente de los ayuntamientos de Madrid, Barcelona y Murcia). Esta aplicación, por tanto, se centra en la **autenticación** con el DNIe [12]. No se ha probado el acceso a portales de administraciones locales al no haber ninguna andaluza, y respecto al resto puede ser útil para múltiples ocasiones, pero la interfaz es muy mejorable.

2.5. Mejoras para las soluciones de empresas privadas

A raíz de lo analizado, se expondrán en primer lugar las ventajas, los inconvenientes y las mejoras posibles de los tipos de soluciones no pertenecientes a la Administración Electrónica

[10] https://play.google.com/store/apps/details?id=es.gob.fnmt.dniesmartconnect
[11] https://play.google.com/store/apps/details?id=es.gob.fnmt.droidpdfsignature.std
[12] https://play.google.com/store/apps/details?id=com.dnieloginwidget

Pública, y finalmente los mismos aspectos, pero de las soluciones de la Administración Española.

- **Soluciones basadas en extensiones:**

  - **Ventajas:** las ventajas principales de las soluciones basadas en extensiones son la **facilidad de uso e integración** con aplicaciones web y nativas, siendo buenas intermediarias entre estas. Además, en el caso de las últimas, no es complejo desarrollar aplicaciones de escritorio que hagan uso del paso de mensajes nativo, requiriendo poca configuración por ambas partes. Por tanto, si definen correctamente las interfaces, y estas a su vez no son complejas de satisfacer, estamos ante una solución que puede ser el **futuro inmediato de la autenticación y la firma en la web**. No es descabellado imaginar una extensión única europea de la que puedan hacer uso todos los países en sus sistemas de administraciones públicas con diversos componentes web y nativos. Además, en su mayoría suelen ser soluciones **gratuitas** (si son de las administraciones, al menos, deberían serlo).
  - **Inconvenientes:** por un lado, **no se tratan de soluciones completas**, sino de un componente en una arquitectura que para funcionar requiere, como se ha comentado, aplicaciones tanto nativas como web que no tienen por qué estar proporcionadas por el mismo desarrollador de la extensión, el cual probablemente se limite a definir las interfaces a implementar. Por otro lado, el mayor problema es **la obsolescencia y la compatibilidad**, pues hoy en día existen un gran número de navegadores cada uno con sus versiones y peculiaridades. Por tanto, se suele dar el caso de haber casi **una versión de la extensión para cada navegador**, o distintos ficheros de configuración de la misma extensión. Esto dificulta el mantenimiento de la extensión en sí y de la infraestructura en general.
  - **Mejoras:** a pesar de que el inconveniente de la compatibilidad sea en muchas ocasiones más cuestión de los propios navegadores, se puede **intentar reducir al mínimo el uso de componentes exclusivos y peculiaridades de los navegadores**, incrementando la comunicación directa y sin pasar por el navegador entre los componentes desarrollados independientemente (aplicaciones nativa, web y extensión). También, como se ha comentado, el éxito de una extensión va a pasar

por la facilidad de incorporación del resto de componentes, por lo que una extensión debería definir **interfaces sencillas de implementar**, e incluso **ejemplos** o directamente plantillas de uso, pues suele ser más sencillo comprender un ejemplo que navegar por una documentación de uso densa.

- **Soluciones hardware:**

  - **Ventajas:** las soluciones hardware mencionadas revelan que probablemente sean el tipo que **ofrece un nivel de seguridad mayor**, pudiendo trabajar hasta con parámetros biométricos. Asimismo, si cuentan con mecanismos de seguimiento, se pueden anular o impedir su uso, como ocurre con las tarjetas bancarias. También son muy interesantes, puesto que potencialmente un dispositivo programable con componentes criptográficos pueden convertirse en soluciones de firma, y si son extensibles modularmente como los Arduino, pueden aportar ideas muy originales.
  - **Inconvenientes:** quizás el menor inconveniente es que sean soluciones **de pago**, pudiendo alcanzar precios bastante elevados si hacen uso de tecnologías sofisticadas. Sin embargo, esto hace que **el extravío** de un token hardware **suponga un coste considerable para la empresa**, y si no cuenta con mecanismos de seguimiento, que conlleve más riesgo su robo o extravío. Además, si no disponen de componentes compatibles adecuados, añadir funcionalidades como el registro de uso puede ser más complejo, y **la elaboración de ejemplos puede ser más complicada** debido a que la interfaz de uso es a menor nivel.
    Otro problema que puede surgir es el de **compatibilidad de controladores**, pues si no se ofrece un correcto mantenimiento, el token se puede quedar obsoleto cuando el sistema operativo o los controladores queden igualmente anticuados, y la elaboración de controladores no es precisamente sencilla.
  - **Mejoras:** una de las soluciones para los token hardware es similar a la de las extensiones, y es dar **ejemplos** de uso o plantillas para integrarlos en todo tipo de aplicaciones, tanto nativas como web incluso con ejemplos de elaboración de drivers compatibles. Finalmente, para evitar o reducir riesgos por extravío o robo, es necesario que cuenten con **mecanismos de seguimiento y anulación**, o incluso de **permisos de uso**, es decir, que el propietario del token sea notificado con

cada intención de uso y manifieste de alguna forma el consentimiento (con otro dispositivo hardware, por ejemplo).

- **Soluciones cloud:**

  - **Ventajas:** en primer lugar, las soluciones cloud permiten **disponer del certificado de firma en cualquier sitio**, pudiendo firmar normalmente o de forma delegada independientemente de dónde se encuentre el firmante, y la **incorporación sencilla** de medidas de seguridad adicionales, como por ejemplo la limitación de acceso, creación de niveles de privilegios, registros de uso, etc. También hacen que la firma **no dependa del dispositivo** en el que se realiza, siempre y cuando tenga alguna forma de participar en el proceso de firma.
  - **Inconvenientes:** las soluciones cloud, sin embargo, tienen la desventaja principal de tener que **confiar en las medidas de seguridad** de los proveedores del entorno cloud tanto a nivel de las aplicaciones proporcionadas como de la infraestructura donde se aloja el certificado, pudiendo verse afectada una gran cantidad de información sensible por un ataque a cualquiera de estos niveles. De la misma manera, también está sujeto en cierta medida a los mecanismos de seguridad del medio de acceso a la plataforma cloud (navegadores, dispositivos móviles, etc.).
  - **Mejoras:** las soluciones principales para mitigar los problemas de los entornos cloud pasan por tener una infraestructura lo más rígida posible ante ataques, pero también es una buena idea **contar con algún tipo de copia de seguridad del certificado fuera del entorno cloud**, para que este no se convierta en el único punto de riesgo.

Como **conclusión**, se puede inferir que las principales mejoras que se pueden dar a cualquier solución de delegación de firma pasan por **reducir el nivel de confianza necesario en cualquiera de los participantes para delegar una firma**, tanto en el proveedor de la solución, como en el delegado mediante la inclusión de **mecanismos de control de acceso** al certificado y **registros de uso** del mismo, por ejemplo. Esto no quiere decir que el delegante no necesite confiar en el delegado, pues la delegación lleva inherentemente un depósito de confianza, sino garantizar que cada firma que se hace cuenta con el consentimiento del

delegante y quede registrada en todo momento la mayor cantidad de información relativa al proceso posible.

2.6. Mejoras para las soluciones de la administración pública

La primera mejora en general sin lugar a dudas es tiene que ver con la **documentación de las soluciones**, pues es ambigua en muchas ocasiones, con múltiples referencias a componentes que están obsoletos o sin soporte. Esta cuestión dificulta enormemente la comprensión y el uso de estas herramientas, en especial en los casos en los que el ciudadano o el empleado de la Administración no tenga experiencia en el ámbito tecnológico. Esto incluye por supuesto realizar una limpieza (ocular o directamente eliminar) de información que no sea de utilidad. En estas mejoras se van a obviar las aplicaciones del DNIe debido a que, por un lado, parecen tener poco soporte y continuidad por parte de las propias administraciones, y por otro lado a que normalmente las firmas con el DNIe las realiza el propietario del DNIe, siendo extraños y menores los casos en los que se delega el DNI.

Comenzando con la **suite @firma**, las mejoras para delegación de firma pasan por los ya mencionados mecanismos de **registro de uso** del certificado, e incluso de gestión de **permisos** de uso, pues las firmas en esta suite se hacen con certificados de clave privada o desde el móvil y no hay formas de distinguir entre firma normal y delegada. Por tanto, y debido a que tanto Autofirma como Firm@Movil pueden ser usadas sin Internet, un registro de uso del certificado **tanto local como remoto** es necesario. Además, y viendo que el Autoscript provoca bastantes problemas (al menos al usarse en Valide, si es que lo utiliza), se podría adoptar otro acercamiento para la comunicación con Autofirma desde la web como podría ser el de **extensiones**, con algo similar a la solución de Web-eID, solo que empleando certificados digitales en lugar de tarjetas inteligentes. Idealmente, una solución podría cubrir tanto el uso de certificados como de smartcards, sustituyendo algunas de las aplicaciones del DNIe. La suite se podría ver muy beneficiada por los mecanismos de control de la delegación de firma, ya que como se menciona en el propio Portal de la Administración Electrónica, va más dirigida a ser usada por las **administraciones**, que pueden utilizar la delegación de firma para agilizar sus numerosos trámites diarios al igual que hacen las empresas privadas, muchas de las cuales ya emplean herramientas mencionadas en el apartado de soluciones privadas. Si la delegación ya

de por sí favorece la agilización de trámites, los mecanismos de control pueden hacerlos más seguros sin añadir mucha más complejidad.

Finalmente, la **infraestructura Cl@ve**. Esta solución **ya posee mecanismos de control de delegación de firma** aunque no se mencionen explícitamente, y es que en los casos en los que es necesario un **OTP** (Cl@ve PIN y trámites de alta seguridad con Cl@ve permanente), hay una confirmación y una verificación en dos pasos que se sigue teniendo que hacer aunque el trámite lo realice un delegado. Por tanto, si un tercero hace un trámite en nombre del registrado en Cl@ve, este último recibirá el OTP y sabrá que se está utilizando su identidad digital en su nombre, pudiendo manifestar explícitamente su consentimiento simplemente comunicándole o no el PIN recibido. No obstante, hay que recordar que una de las formas de autenticación en Cl@ve es con **usuario, DNI y contraseña**, que muchas administraciones permiten el uso de certificados instalados en el navegador, y que no todos los trámites de firma usan un OTP, por lo que no siempre es posible controlar el uso de nuestra identidad digital, siendo también necesario por tanto un **registro de actividad**, ya que no se menciona la existencia del mismo, que pueda ser accesible por el propietario de las credenciales o del certificado de autenticación y firma.

En resumen, todas las soluciones de firma de la Administración Electrónica Pública se verían beneficiadas enormemente por un **registro de actividad y uso de certificados consultables por el delegante**, y por un mecanismo de establecimiento de **permisos**. Un buen paso inicial es **fomentar el uso de Cl@ve PIN** en lugar de permitir que los delegados instalen los certificados de los delegantes, aunque para la realización de muchos trámites en poco tiempo tiene la desventaja de ser más lento (entre que el delegante recibe el PIN y se lo comunica al delegado), perdiendo esa agilidad que se obtenía al delegar la firma. Precisamente quizás sea esta una de las razones por las que se sigue pudiendo usar los certificados instalados en muchos portales.

El escenario ideal es aquel en el que los trámites sean más seguros, sin perder agilidad.

3
Metodología

La metodología de desarrollo tendrá como primer objetivo la documentación e investigación sobre la situación actual del proceso de delegación de firma en el ámbito europeo y español, y posteriormente el proceso para desarrollar la prueba de concepto que aplique los conocimientos adquiridos en el apartado previo. Las metodologías concretas de desarrollo del software se aplicarán en la fase del desarrollo de la prueba.
El desglose de las fases es el siguiente:

1. **Investigación previa**:

   - Estudio de la **legislación** actual a nivel nacional y europeo.
   - Análisis de las **soluciones actuales de identidad digital** tanto del sector privado como del sector público.
   - Ventajas, inconvenientes y mejoras de dichas soluciones para asegurar la delegación de identidad digital.

2. **Desarrollo de la prueba de concepto** en las siguientes fases:

   a) **Captura** de requisitos

   b) **Análisis** de requisitos

   c) **Diseño** de la solución, pues hay que valorar varias alternativas hasta encontrar la más adecuada para las condiciones de este trabajo, especialmente en lo que a viabilidad de tiempo se refiere.

   d) **Implementación** en forma de iteraciones tanto de las funcionalidades como de los requisitos no funcionales de la solución.

e) Pruebas. Si se dispone de tiempo suficiente, se desarrollarán pruebas más formales. Nótese que no se han seguido metodologías de desarrollo como Test Driven Development.

La fase de **implementación** de la prueba de concepto se hará, como se ha mencionado, **en iteraciones** para implementar funcionalidades concretas o características no funcionales. Las iteraciones tendrán una de **extensión variable** en vista a los problemas que se vayan encontrando, pues se trata de un territorio poco explorado. No obstante, un **máximo razonable** de duración de la iteración que impida el bloqueo del desarrollo será de **tres semanas**, pasando al desarrollo de otros componentes o partes de la solución si en ese plazo no se ha cumplido el objetivo propuesto.

Téngase en cuenta que el principal propósito de este TFG no implica el desarrollo completo de un producto software, sino la proposición de soluciones para asegurar el proceso de delegación de identidad digital e investigar en qué medida se pueden llevar a cabo en los plazos del trabajo. Además, para **facilitar la continuidad del trabajo realizado**, se irá documentando a nivel algo más técnico las dificultades encontradas en el proceso en uno de los anexos de este trabajo.

4

Captura y análisis de requisitos

Para la captura de requisitos de la prueba de concepto, la mejor documentación disponible ha sido la investigación realizada previamente.

Los requisitos de la solución de delegación de firma no se centran tanto en las funcionalidades que ofrece, pues no son numerosas, sino en los requisitos no funcionales, ya que de lo que se trata es de mejorar el proceso en términos de seguridad y comodidad, además de ofrecer funcionalidades no ejecutadas directamente por los usuarios, sino por el propio servicio, entre otros.

Esta sección se detallan los requisitos de la solución de delegación de firma en sí, esto es, **los requisitos de la combinación del servidor de registros y las funcionalidades añadidas tanto a la aplicación nativa modificada como a la extensión modificada**.

4.1. Requisitos funcionales

- **RF01. Autenticación:** El usuario podrá acceder al servidor de registros mediante la autenticación con certificado.

- **RF02. Consulta de registros:** El usuario podrá visualizar una lista con detalles de los registros de uso de su certificado.

- **RF03. Descarga de registros:** El usuario podrá descargar los registros desde el servidor de registros, pues puede perder los almacenados en local.

- **RF04. Descarga de documentos firmados:** El usuario podrá descargar desde el servidor de registros el documento asociado a cada registro de firma.

4.2. Requisitos no funcionales

- **RNF01. Usabilidad:** La solución propuesta ha de ser lo más sencilla posible de usar, requiriendo por parte del usuario el menor número de acciones posibles, y que estas sean intuitivas.

- **RNF02. Registro local y remoto**: La solución llevará un **registro** tanto local (máquina en la que se realiza) como remoto (en el servidor de registros) de los usos del certificado. De esta forma no existe un único punto de error y la pérdida de datos en un punto no significa la pérdida total.

- **RNF03. Contenido de los registros**: Los registros contarán con campos que permitan identificar lo máximo posible la actividad. Como mínimo serán:

  - **Ip pública** desde la que se realizó la actividad.
  - **Fecha** de la actividad.
  - **Tipo de máquina** desde la que se realizó la actividad (plataforma).
  - **URL origen** desde la que se realizó la actividad.
  - **Identificadores** de los datos enviados desde la página web (hash del documento que se firma, nonces de tokens de atenticación, etc.)
  - **Tipo de actividad** (autenticación o firma).
  - **Estado** de la actividad, es decir, indicar si la acción de autenticación o firma se completó con éxito o quedó en un paso intermedio.

- **RNF04.Tiempo de respuesta:** Siendo conscientes de que la satisfacción de los requisitos de seguridad pueden conllevar tiempos de espera adicionales, los tiempos de respuesta deberán ser lo más bajos posible.

- **RNF05.Protección de la base de datos:** Los registros de actividad solo podrán ser accedidos desde el servidor, de forma que a la hora de hacer modificaciones en la base de datos siempre se pasará por el servidor.

- **RNF06. Protección de los puntos de acceso del servidor:** Sólo se podrán acceder a los endpoints del servidor si se satisfacen determinadas condiciones de seguridad, como pueden ser:
  - Incluir campos o elementos de autenticación junto con las peticiones (tokens, nonces…).
  - Reducir al mínimo el número de endpoints del servidor para reducir en consecuencia los puntos de posibles ataques.
  - Imposibilitar determinadas acciones si no se ha iniciado sesión en el servidor.
- **RNF07. Acceso desde sitios seguros**: La solución sólo permitirá la autenticación y firma desde sitios web seguros (https).

5 Diseño

Para el diseño de la prueba de concepto, se han realizado varios acercamientos previos a la decisión final, pues al tratarse de una prueba de concepto, **se han investigado paralelamente varias alternativas** y valorado su viabilidad para determinar si la meta se podía alcanzar con mayores o menores dificultades con cada una. Señalar asimismo que incluso al tomar una decisión de diseño aparentemente final, no hay garantía completa de que fuera a llevar al objetivo esperado en los plazos previstos, desembocando en modificaciones del diseño inicial.

5.1. Primeros acercamientos

Las primeras soluciones de diseño barajadas han sido:

1. **Navegador portable con extensión:** consistiría en que el usuario, al delegar su certificado a un asesor, por ejemplo, lo hiciera mediante la entrega de una memoria USB que contenga una versión portable de un navegador con el certificado del usuario instalado en él. Este navegador contaría con una extensión habilitada cuya funcionalidad fuera reconocer los sitios en los que se hace un uso de los certificados instalados y guardase un registro de esta actividad tanto en la memoria como en el servidor de registros. La forma de almacenar este registro podría incluir junto a los registros capturas de pantalla del momento en el que se hace uso del certificado.
   Se trata por tanto de proporcionar un **entorno seguro y aislado para usar el certificado**, combinando el diseño basado en extensiones y hardware. Sin embargo, la desventaja lógica es la **gestión de numerosas memorias USB** por parte del asesor.

2. **Solución hardware:** ya que se tenían a disposición dispositivos programables como el Nordic Thingy:91[13], que además cuenta no solo con un chip criptográfico, sino también

[13] https://www.nordicsemi.com/Software-and-tools/Prototyping-platforms/Nordic-Thingy-91

con compatibilidad con BLE (Bluetooth Low Energy) y NFC, se barajó la posibilidad de utilizarlo de varias formas.

La primera idea base consistió en algo similar a la propuesta anterior, solo que **sustituyendo la memoria USB por este dispositivo**, que también almacenaría los certificados que se delegan. Gracias al BLE se podría utilizar el certificado en distintas máquinas sin necesidad de tener que instalar cada certificado en un navegador ni conectar el dispositivo a cada una, incrementando la comodidad y permitiendo medidas de seguridad adicionales como la confirmación en dos pasos.

No obstante, y esta es la segunda forma que se planteó, se podría **prescindir de la extensión** para elaborar una solución más centrada en hardware, situando en el dispositivo el control de uso y mandando los registros al servidor desde él. Por tanto, esta solución es un lector NFC modificado, o un **token hardware de firma con vigilante online en tiempo real**, pues el dispositivo tiene integrados los certificados y se comunicaría con un servidor que almacenase los registros y que fuera accesible por el usuario, y para ello, se podría emplear algún estándar como OpenSC. Si en ese servidor, además, están almacenados los certificado del usuario, sería una combinación de solución cloud y hardware. Podría además añadirse el hecho de que una copia del certificado dependiera de la otra, es decir, que al eliminarse este del hardware, no funcionase el de la nube y viceversa.

En todas estas propuestas, el servidor de registros sería una aplicación web que funcionaría como una **api REST**.

Es en esta fase del trabajo en las que se han empleado tecnologías mencionadas en la introducción de menor nivel como la NRF Connect Suite y Segger Studio (que incorpora Zephyr[14]), además claro está del propio dispositivo, para el estudio de esta alternativa.

Sin embargo, y aunque se estudiaron las formas de uso del Thingy:91, no se ha acabado trabajando con él al investigar y encontrar paralelamente otros acercamientos interesantes más inmediatos, sin requerir una adaptación del dispositivo a OpenSC ni la posible necesidad de elaborar un driver, pues OpenSC está expresamente pensado para tarjetas inteligentes.

[14] https://zephyrproject.org/

5.2. Simulación de tarjetas inteligentes

Antes de trabajar con el dispositivo hardware directamente y entrar en la posible implementación de un driver o de un mecanismo de comunicación adaptado a OpenSC, se valoró realizar todo el proceso mediante la simulación de tarjetas inteligentes. De esta forma, la comunicación vía OpenSC ya estaba cubierta, e implicaba que **si la solución funcionaba con la tarjeta simulada, se podría aplicar a tarjetas reales y potencialmente a ficheros de certificados**. Para esta propuesta se estudiaron tanto las alternativas de simulación de tarjetas inteligentes de la sección de *wiki* del propio proyecto OpenSC como el proyecto Vsmartcard[15].

5.2.1. Vsmartcard

El autor de este proyecto es uno de los colaboradores del propio proyecto OpenSC y ha desarrollado una arquitectura con varios componentes [44]:

- **Virtual Smart Card:** para crear tarjetas inteligentes virtuales y hacerlas accesibles mediante el mecanismo de comunicación PC/SC[16].
- **Remote Smart Card Reader:** permite a un ordenador utilizar el sensor NFC de un smartphone como lector de tarjetas inteligentes.
- **Android Smart Card Emulator:** permite utilizar un smartphone Android como tarjeta inteligente sin contacto mediante la emulación de esta (similar a la forma de hacer pagos con el móvil en tiendas físicas) y haciendo uso del Host Card Emulator (HCE) de Android[17].
- **Tizen Smart Card Emulator:** similar al anterior, utilizando el HCE de Tizen[18] en lugar del de Android.
- **PC/SC Relay:** permite la comunicación entre una tarjeta inteligente y una interfaz sin contacto (como puede ser un datáfono), mediante la transmisión de comandos APDU[19]

[15] https://frankmorgner.github.io/vsmartcard/index.html
[16] https://es.wikipedia.org/wiki/PC/SC
[17] https://developer.android.com/guide/topics/connectivity/nfc/hce
[18] https://developer.tizen.org/community/tip-tech/tizen-nfc-card-emulation-mode
[19] https://en.wikipedia.org/wiki/Smart_card_application_protocol_data_unit

entre ambas partes. De este componente se sirven las tarjetas virtuales simuladas mencionadas anteriormente.

- **USB CCID Emulator:** permite que un lector de tarjetas inteligentes con soporte PC/SC funcione como un lector CCID USB, pudiendo servir de intermediario de confianza que asegure modificación e introducción de PINs segura.

De esta arquitectura, interesa inicialmente **la simulación en ordenador de tarjetas inteligentes**, es decir, el componente Virtual Smart Card. El funcionamiento de este componente consta de varias partes:

- Una **aplicación que utiliza las tarjetas inteligentes**, normalmente una aplicación nativa. No se menciona ningún lenguaje de desarrollo determinado.
- El **framework PC/SC**: proporciona una API que pueden emplear las aplicaciones que utilizan las tarjetas inteligentes para comunicarse con los lectores de tarjetas.
- **VPCD:** se trata de un **driver** para hacer que la tarjeta inteligente sea accesible vía PCSC-Lite y el servicio de tarjetas inteligentes de Windows. Este driver está implementado en lenguaje C.
- **VPICC**: la **implementación de la tarjeta inteligente virtual**. Está desarrollada en lenguaje Python.

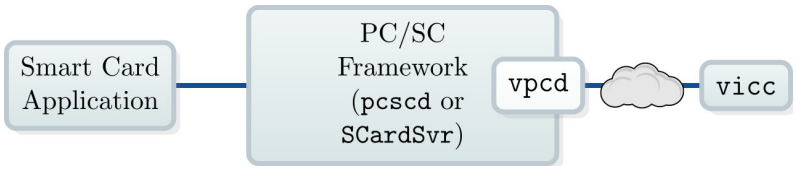

Figura 1: Arquitectura de Virtual Smart Card utilizada con PCSC-Lite o WinSCard [45]

La última versión trae además una implementación de PC/SC propia, de forma que no es necesario el uso de PCSC-Lite, reduciéndose la solución al siguiente esquema.

Los **problemas** encontrados pasan principalmente por el **mantenimiento** general del proyecto y las **compatibilidades**, principalmente en Windows, que es el sistema operativo en el que se ha desarrollado este trabajo.

Figura 2: Arquitectura de Virtual Smart Card utilizada con implementación propia de PC/SC [45]

Para comenzar, el funcionamiento del driver se basa en extender un proyecto de un driver para lectores de tarjetas inteligentes virtuales (BixVreader) que **data de 2014** [20]. A eso hay que sumar que en ese proyecto se menciona que solo se ha probado que funcione correctamente en sistemas con **Windows 7 de 64 bits**.
Otro punto problemático es el **uso del WDK** (Windows Driver Kit) en el proceso, pues no solo es hay diferencias entre los WDK de Windows 10 y el de Windows 7, sino que hay distintas versiones del mismo dentro del propio Windows 10. Respecto a las **herramientas** del proyecto, se hace uso de Visual Studio 2015, y versiones antiguas de otras. A esto hay que sumarle el uso de **Python 2.7** (versión muy antigua) y de la librería PyCrypto de Python, que también está obsoleta y prácticamente sustituida por completo por PyCryptodome y Cryptography.

Todos estos aspectos han desembocado en la imposibilidad de, en primer lugar, compilar con éxito el driver BixVReader tanto en Visual Studio 2019 como en Visual Studio 2015, y en segundo lugar, poder hacer funcionar las tarjetas virtuales (vicc). Más detalles sobre los problemas en esta fase se sitúan en el anexo de problemas encontrados.

5.2.2. Soluciones de OpenSC

En la búsqueda de alternativas similares a la propuesta anterior, resulta que OpenSC cuenta en su *wiki* con una sección dedicada a la simulación de tarjetas virtuales[21]. No obstante y desafortunadamente, **se hace referencia al mismo driver de 2014 que causó la mayor parte de los problemas anteriores**. Es cierto que se ofrecen más detalles de su uso, y en este caso la simulación de tarjetas inteligentes se hace a través de varias applets de Java, pero el problema del lado del driver continuaba presente. Por tanto, aunque se pudiera simular la tarjeta correctamente con alguna de estas applets, no sería posible simular la comunicación

[20] https://www.codeproject.com/Articles/134010/An-UMDF-Driver-for-a-Virtual-Smart-Card-Reader
[21] https://github.com/OpenSC/OpenSC/wiki/Smart-Card-Simulation

con un lector. Más detalles técnicos de los problemas se encuentran en los anexos, pues es importante mencionarlos para posibles investigaciones futuras que partan de este trabajo.

Por tanto, en vista de que las principales opciones de simulación de tarjetas virtuales pasan por un driver antiguo y de soporte y compatibilidad dudosas, **se decidió optar por investigar el enfoque de las extensiones.**

5.3. Web-eID

La iniciativa de web-eid tiene el principal objetivo de permitir la **autenticación y firma en la web con tarjetas inteligentes de identidad de la Unión Europea (eID)**. Partiendo de proyectos anteriores como Open eID, intenta hacer más cómodo el uso de los eIDs en la web, aspirando a permitir en un futuro el uso de todas las tarjetas, pues actualmente son pocas las compatibles. Realmente, esta nueva arquitectura surge como respuesta a los problemas que ahora afronta el proyecto Open eID causado principalmente por el acoplamiento con con los navegadores [46].

La solución consta de una **librería de JavaScript**, una **extensión** y una **aplicación nativa**, permitiendo la comunicación sencilla entre el navegador, la aplicación web y la tarjeta inteligente [46].

La implementación de Open eID actual es muy dependiente de la plataforma, con muchos componentes únicos para cada navegador, lo cual permite mejor integración con ellos, pero tiene **grandes problemas de mantenimiento** tanto de APIs y comunicación entre componentes como de los propios componentes en sí por los cambios y la evolución de los navegadores, los protocolos y los sistemas operativos, siendo una solución demasiado compleja.

Ante esta situación, han optado por un acercamiento enfocado en **desacoplar** la solución de todas las plataformas, es decir, del navegador, para lo cual se utilizarán las **extensiones** y del sistema operativo, para lo que se utilizará la **API PC/SC** (que ha aparecido ya anteriormente en las soluciones de diseño) para la comunicación directa con la tarjeta, pues es también multiplataforma [46].

Otros principios consisten en permitir solo sitios seguros (*https*), no cachear los PIN, emplear procesos similares para autenticación y firma y ofrecer soporte para más tarjetas inteligentes de identidad europeas (como el DNI) [46]. La arquitectura de la solución se reduce,

por tanto, a los siguientes componentes:

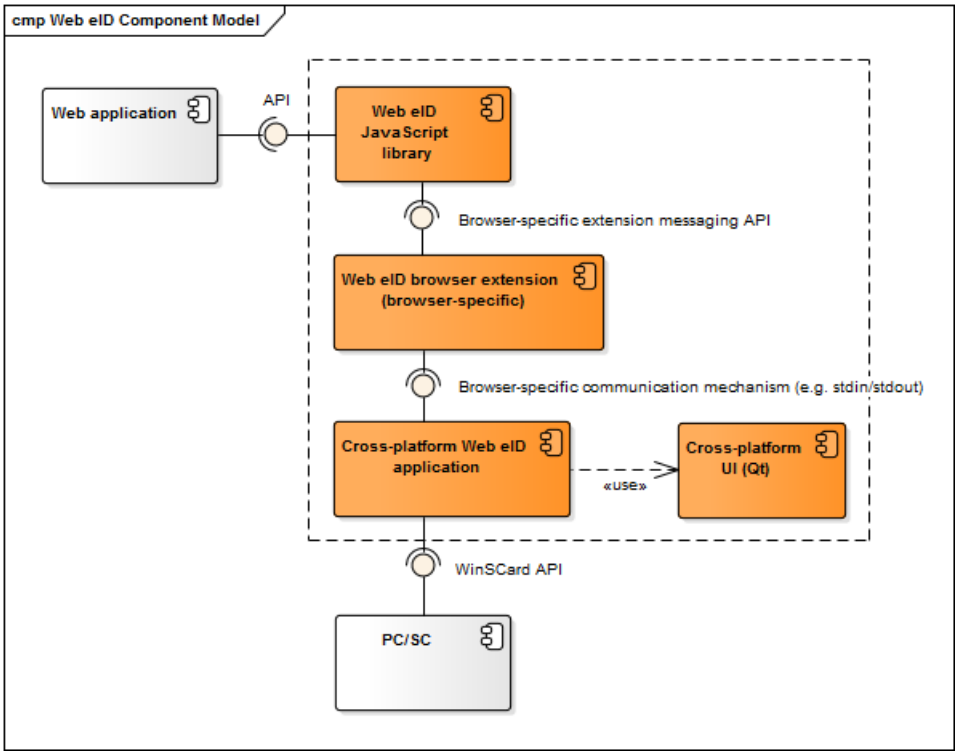

Figura 3: Diagrama de componentes de la solución de Web-eID [46]

Con esta arquitectura, ya se satisfacen otros requisitos que se venían persiguiendo, como funcionar con una variedad importante de tarjetas de identidad (Estonia, Letonia, Lituania y Finlandia), con las últimas versiones de los principales navegadores, tener una **fácil instalación**, sin software de terceros (lo cual permite una solución enteramente pública) y sin interferir con otros componentes de otras aplicaciones centrados en la identidad digital [46].

El flujo de funcionamiento de la solución se basa en paso de mensajes entre los componentes, lo cual permite que los procesos, aunque secuenciales, sean asíncronos y permitan actuar ante una serie de posibles acciones como la retirada de la tarjeta inteligente del lector.

A continuación se describen los **flujos de autenticación y firma**, pues se utilizarán como base y continuarán lo menos alterados posibles durante la modificación que será la prueba de concepto. Señalar que cuando se menciona la *librería de Javascript*, se hace referencia al *frontend* de la aplicación web, pues es donde se debe situar la librería para satisfacer la estructura de la solución. Esto permite que las acciones del usuario *despierten* tanto al servidor de la aplicación web como a la extensión, que será la que se comunique con la aplicación nativa.

Para la **autenticación, los pasos se describen en el siguiente diagrama y son [46]:**

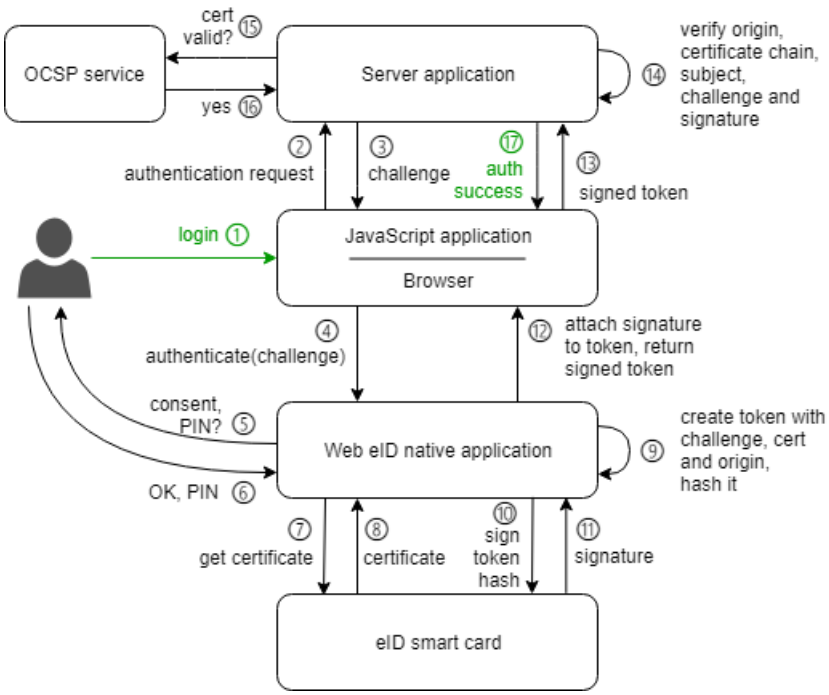

Figura 4: Flujo de autenticación con Web-eID [46]

1. El usuario inicia el inicio de sesión.

2. La librería de JavaScript, situada en el *frontend* de la aplicación web manda la petición al servidor de la aplicación web.

3. El servidor responde con el **challenge**, es decir, con el nonce que él genera.

4. La librería de JavaScript ejecuta el método *authenticate(challenge)* de web-eid.js, que hace que la extensión automáticamente lance la aplicación nativa pasándole el nonce, la URL origen y opcionalmente otros parámetros.

5. La aplicación nativa pide el consentimiento y el PIN al usuario mediante interfaz gráfica.

6. El usuario da el consentimiento e introduce el PIN.

7. La aplicación nativa intercambia comandos APDU con la tarjeta para obtener el certificado de autenticación. Se hace mediante la API PC/SC.

8. La tarjeta responde con el certificado (en forma de respuestas APDU con el certificado en bytes).

9. La aplicación nativa crea el token (un token jwt que sigue el estándar OpenID X509 ID Token) con el certificado, los argumentos pasados por la extensión y el nonce, para luego hacer un hash de todo.

10. La aplicación nativa se intercambia comandos APDU con la tarjeta para que esta firme el hash, previa verificación del PIN.

11. La tarjeta responde con la firma del hash del token.

12. La aplicación nativa manda el token firmado a la extensión.

13. La extensión manda el token firmado al servidor de la aplicación web.

14. El servidor verifica el certificado y el token, validando la dirección web (en el campo *aud* del token), el nonce (comparándolo con el que generó) y otros aspectos como el Token Binding Hash (*tbh*), el sujeto, la cadena de confianza del certificado y la firma del token (pasando el token y el certificado).

15. El servidor comprueba el estado del certificado (petición OCSP).

16. La aplicación responde diciendo que el usuario está autenticado.

Cabe mencionar que esta solución **protege ante ataques *man-in-the-middle* y ataques de reproducción de tokens de autenticación** al no permitir que ninguna de las peticiones se haga desde otro origen que no sea el del inicio el proceso, y además este origen se valida[22].

Como se ha mencionado, **el proceso de firma se pretende que sea similar** en esta solución, y se va a basar en lo mismo: que se firmen en la parte de la aplicación nativa unos datos enviados por el servidor utilizando la clave privada de la tarjeta inteligente. La firma se devuelve al servidor y verifica la validez del certificado y la firma con la clave pública del certificado. Concretamente **el esquema y los pasos de la firma son los siguientes [46]:**

[22] https://github.com/web-eid/web-eid-system-architecture-doc#protection-against-man-in-the-middle-attacks-during-authentication-with-origin-validation-and-tls-token-binding

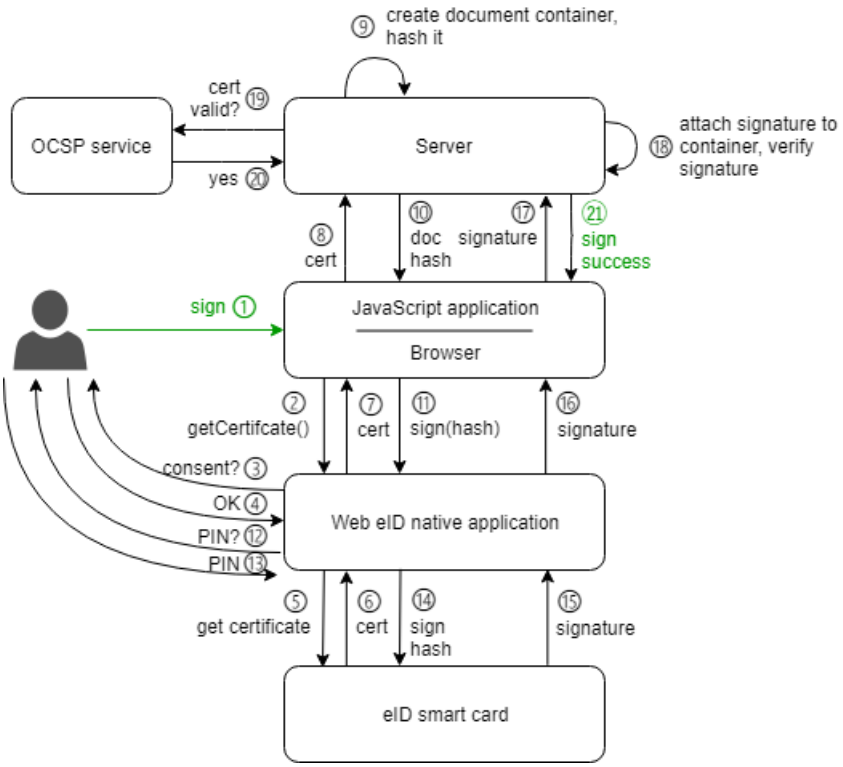

Figura 5: Flujo de firma con Web-eID [46]

1. El usuario inicia el proceso.

2. La librería de JavaScript ejecuta el método *getCertificate*, lo que hace que la extensión lance la aplicación nativa con el comando *certificate*.

3. La aplicación nativa pide el consentimiento al usuario mediante interfaz gráfica para mandar su certificado de clave pública al servidor. Esto es necesario ya que el certificado contiene datos personales.

4. El usuario da su consentimiento.

5. La aplicación nativa intercambia comandos APDU vía PC/SC para seleccionar el certificado, obtenerlo y leer su contenido.

6. La tarjeta responde con los bytes del certificado vía APDU.

7. La aplicación nativa manda el certificado de firma de vuelta a la extensión.

8. La extensión manda el certificado al servidor de la aplicación web.

9. El servidor crea el contenedor de firma (siguiendo el estándar Associated Signature Containers (ASiC) empleando por ejemplo librerías existentes como DigiDoc4j), añade el documento a firmar y y calcula el hash que se firmará en el lado de la aplicación nativa.
10. El servidor manda el hash a la librería de JavaScript.
11. La librería de JavaScript ejecuta el método *sign(certificate, hash)* y la extensión manda el comando *sign* a la aplicación nativa, con el certificado y el hash como argumentos.
12. La aplicación nativa vuelve a pedir el PIN al usuario con una interfaz gráfica que también describe los datos a firmar.
13. El usuario introduce el PIN.
14. La aplicación nativa intercambia comandos APDU con la tarjeta para firmar el hash previa verificación del PIN.
15. La tarjeta responde con la firma del hash realizada con la clave privada y responde vía comandos APDU con ella.
16. La aplicación nativa manda la firma a la librería de JavaScript.
17. La librería de JavaScript manda la firma al servidor.
18. El servidor añade la firma al contenedor y lo valida.
19. El servidor manda el certificado a servidor OCSP.
20. El servidor confirma que se ha realizado la firma correctamente

En base a esta arquitectura, se han pensado **tres alternativas de modificación** para ofrecer mayor soporte y seguridad de delegación de identidad digital, que se ordenan a continuación según viabilidad.

1. **Reemplazar la aplicación nativa**: desarrollar una nueva aplicación nativa (por ejemplo en Python) **que trabaje con ficheros de certificados** en lugar de con tarjetas físicas, sin alterar en la medida de lo posible el resto de componentes. La extensión, por ejemplo, simplemente tendría que apuntar a la nueva aplicación nativa, y el resto se mantendría. Para el desarrollo de la nueva aplicación nativa:

- Hay que desarrollarla en base al flujo de eventos, mensajes y formatos de la extensión y de la librería de JavaScript.
- Se pueden emplear librerías criptográficas de Python como Pycryotodome o Cryptography para realizar las operaciones que haría la tarjeta inteligente.
- Hay que añadir nuevos pasos y/o modificar algunos para soportar la delegación de firma mediante la comunicación con el servidor de registros y el almacenaje de registros en local.

2. **Fork de la aplicación nativa nativa**: **modificar** la aplicación nativa existente, en lenguaje C++, eliminando o inhabilitando la conexión con smartcards y sustituyéndola por el uso de ficheros de certificados. Para estas modificaciones:

   - Hay que implementar las operaciones criptográficas sin la tarjeta inteligente, para lo cual se pueden emplear librerías criptográficas de C++.
   - Al igual que en el paso anterior, hay que añadir nuevos pasos o modificar algunos para registrar el uso del certificado en el entorno local y en el servidor de registros.
   - La **compilación y depuración** del proceso es más costosa, al ser un proyecto de gran envergadura en lugar de una aplicación que se desarrolla desde cero y va creciendo.

3. **Interceptar las operaciones de la tarjeta y añadir soporte de delegación de firma**.

   - Modificar la aplicación nativa existente para modificar directamente los pasos. Es decir, se seguiría trabajando con tarjetas inteligentes.
   - Disponer de tarjetas inteligentes actualmente compatibles con Web-eID y lectores soportados.

   Esta opción no obstante, aparte de ser la menos viable en términos de recursos, es la que limita más la funcionalidad, reduciéndola a las tarjetas inteligentes actualmente soportadas, y no a los ficheros de certificados. Otra opción dentro de esta solución de diseño sería añadir soporte para DNIe, pero implicaría conocer todos los detalles a bajo nivel necesarios.

A partir de la siguiente sección, se describe cómo se ha ido desarrollando la prueba de concepto con la primera de las alternativas mencionadas. El primer paso, por tanto, será **adaptar los flujos de autenticación y firma descritos a la solución con la nueva aplicación nativa y con ficheros de certificados**. En segundo lugar, finalmente, **se extenderá y/o modificarán estos flujos para añadir el soporte de delegación de firma**. Más detalles sobre la arquitectura de la solución base de Web-eID se pueden encontrar en el repositorio de GitHub dedicado a ello [46].

6

Fase 1. Adaptación de la solución de Web-eID

6.1. Requisitos

Los requisitos de lo desarrollado (más bien modificado) en esta fase son principalmente las funcionalidades de la aplicación existente, pero empleando certificados españoles en lugar de tarjetas inteligentes.

Requisitos funcionales

- **RF01.Autenticación:** El usuario podrá autenticarse con su certificado de autenticación.
- **RF02.Firma:** El usuario podrá firmar documentos haciendo uso de sus certificados de firma.

Requisitos no funcionales

- **RNF01.Tipo de certificados aceptados:** La modificación de la solución de Web-eID permitirá como mínimo el uso de certificados españoles emitidos tanto por la AC-FNMT Usuarios (ACUSU) por la Gerencia Informática de la Seguridad Social (GISS).
- **RNF02.Tiempo de respuesta:** la versión modificada mantendrá tiempos de respuesta similares o menores a los que tenía originalmente, sin contar el tiempo adicional empleado una vez se incorpore la solución de delegación de firma.

6.2. Iteración 0. Comunicación

Esta fase se ha considerado como fase cero al no añadir funcionalidades, pero ser importante para determinar si el enfoque de extensiones es viable para la prueba de concepto. Se comprobará que es efectivamente posible establecer la comunicación entre la extensión y una aplicación nativa distinta de la original, en este caso la desarrollada desde cero en Python. Esta comprobación se hará directamente con la web de prueba de Web-eID[23] y la depuración de extensiones de Firefox. Otra fuente de documentación ha sido la propia librería *web-eid.js*[24].

Para esta fase, los pasos serán por un lado **compilar la extensión** de nuevo **haciendo que apunte a la nueva aplicación nativa** y crear la **primera versión** de dicha aplicación nativa que solo responda al primer evento del proceso.

En el **primer paso** solo cabe señalar que se ha modificado uno de los archivos de compilación (ver el anexo de problemas) para que el proceso se completase con éxito.

Para comenzar con la aplicación nativa, se han utilizado como base por un lado, **el ejemplo de paso de mensajes nativo en Firefox**[25] y por otro, **la propia aplicación nativa original de Web-eID**, con el objetivo de conocer el formato de los mensajes entre los componentes y de esta forma replicarlo en Python sin modificaciones innecesarias.

Para activar el proceso de autenticación en la página de ejemplo basta pulsar el botón de *Authenticate*. El siguiente paso es responder al evento, pues ni la aplicación web de ejemplo ni la extensión envían nada, sino que esperan concretamente un mensaje con de la aplicación nativa. Si se envía, por ejemplo "python", la respuesta es la siguiente:

Figura 6: Respuesta de la extensión con mensaje distinto de la versión

Por tanto, el formato del mensaje de versión debe ser un JSON con la estructura *{ "version":*

[23] https://web-eid.eu/

[24] https://github.com/web-eid/web-eid.js

[25] https://github.com/mdn/webextensions-examples/tree/master/native-messaging

"x.y.z"}.

De ahora en adelante, en las iteraciones describirán los principales problemas encontrados a alto nivel y las soluciones. En esta fase, estos han sido:

- **Problema**: al codificar el mensaje con la versión con el método proporcionado en el ejemplo de MDN, se obtiene un error al ser el mensaje demasiado grande.
- **Solución**: en el proyecto de la aplicación nativa original hay unos tests para la clase *inputoutputmode* en Python con una forma distinta de codificar el mensaje[26]. Con dicho método se ha resuelto el problema, consiguiendo mandar la versión y obteniendo una respuesta de la extensión.

Figura 7: Respuesta exitosa de la comunicación con la extensión.

Como se aprecia, la respuesta contiene un comando, el nonce generado por el servidor de la aplicación y el certificado de esta. El formato de mensaje {comando:comando, argumentos: []} es el que se utiliza en todas las transmisiones desde la extensión hacia la aplicación nativa, pero no en sentido contrario.

No obstante, **aún no se ha leído nada de la extensión** en la aplicación nativa, por lo que **no se puede decir que se haya cubierto toda la comunicación entre los componentes**.

[26] https://github.com/web-eid/web-eid-app/blob/main/tests/input-output-mode/test.py

A pesar de que lo que se recibirá ya pertenece al proceso de autenticación, se cubrirá almacenando de alguna forma lo que la extensión envía. Esto permitirá tener implementada la lectura y escritura en la aplicación nativa, pudiendo reutilizarla en el resto de fases. Para la lectura, de nuevo ha sido necesario acudir a los tests de la aplicación nativa.

En esta parte del desarrollo, se ha descubierto que **es complejo depurar la aplicación nativa en ejecución** al ser ejecutada por la extensión y no por el usuario, y para ello se han implementado funcionalidades para detectar y almacenar errores en un fichero de error, y para almacenar el contenido enviado por la aplicación nativa. La secuencia de esta fase es la siguiente:

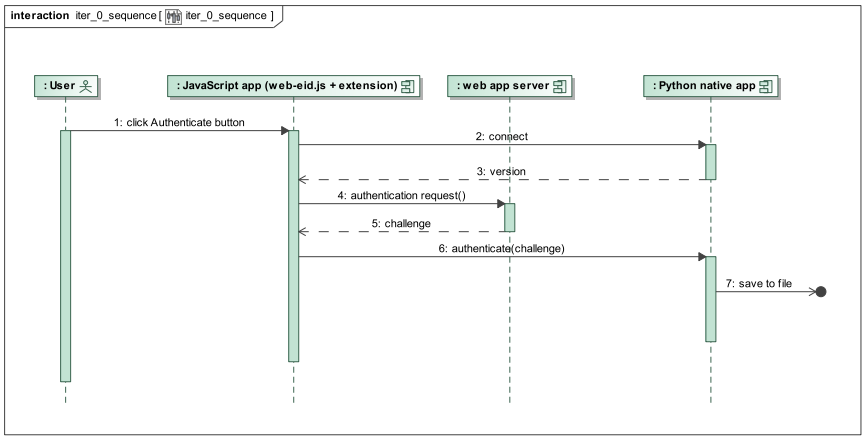

Figura 8: Diagrama de secuencia de la fase de comunicación.

Esta versión temprana de la aplicación nativa, en conclusión, implementa algunas funcionalidades de la clase *inputoutputmode* sin la parte de funcionar en línea de comandos. En las siguientes fases, aparte de cubrir las funcionalidades, se irá refactorizando y modularizando la aplicación nativa para que quede con una estructura similar a la de la aplicación original.

6.3. Iteración 1. Autenticación

Una vez verificado que es posible la comunicación entre la aplicación web de prueba, la extensión mínimamente modificada y la nueva aplicación nativa, se cubrirá la primera funcionalidad: la autenticación. El esquema del flujo de autenticación original se encuentra en la sección de diseño de la prueba de concepto 4 .

El paso inicial de *despertar a la aplicación nativa*, que esta envíe la versión, y luego reciba el mensaje con el comando *authenticate* ya se ha realizado, solo que se ha escrito en un archivo de texto.

La estructura original de la aplicación nativa cuenta con un **controlador** y unos **manejadores de comandos** (*command handlers*), estructura que se imitará para una *traducción* más sencilla, separando también las funciones relacionadas con la comunicación con la extensión en *inputOutput.py*.

En esta iteración, la funcionalidad del manejador original *authenticate* consiste en:

1. **Validar** el formato y el contenido del mensaje recibido de la extensión.
2. Pedir el **PIN** al usuario.
3. Comenzar la **comunicación** vía comandos APDU **con la tarjeta** para obtener el certificado previa comprobación de que el PIN es correcto.
4. **Construir y firmar el token jwt**. De acuerdo al estándar *OpenID X509 ID Token* y tomando los argumentos de *origin*, *nonce* y *subject* tanto del certificado como de los argumentos enviados por la aplicación nativa.
5. **Enviar el token** con la firma de este a la extensión.

Por tanto, lo que se trata de hacer en esta fase es traducir el proceso, **sustituyendo la comunicación con comandos APDU por una lectura del certificado almacenado**. Asimismo, la introducción y comprobación del PIN de momento no se implementará, sino que se el PIN estará en el código directamente, dejándose como una modificación de baja prioridad para fases futuras. La comprobación de errores también será de menor prioridad, pues al trabajar con página de web-eid directamente se supone que son correctos los contenidos enviados. El diagrama de la autenticación de la solución modificada (sin aspectos de delegación de firma aún) es el siguiente:

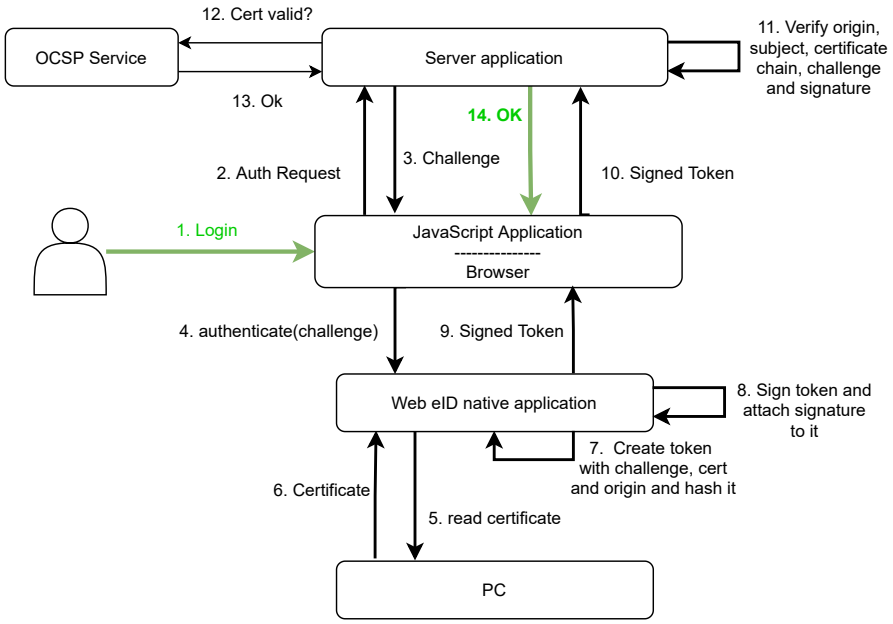

Figura 9: Diagrama de flujo de autenticación de la solución modificada en la fase 1.

Los **problemas y soluciones** en esta iteración han sido:

- **Problema 1: Los certificados españoles no son aceptados por la página de test de web-eid.**
- **Solución:** como se menciona en el repositorio de la librería de validación de tokens de Web-eID[27], se deben almacenar los certificados de las Autoridades de Certificación emisoras correspondientes en las propias aplicaciones web para que se tengan en cuenta a la hora de validar los certificados que emiten. Eso implica que ya **no es posible utilizar la página de test de web-eid, sino que hay que levantar la versión del ejemplo en entorno local** e incluir en ella los certificados de las Autoridades de Certificación de España.
- **Problema 2: los certificados del set de certificados de prueba del DNIe han expirado.** Estos eran los que se pensaba utilizar durante el desarrollo de la prueba de concepto.
- **Solución:** en vistas de que no era posible conseguir certificados de clave privada de prueba de Estonia ni de España (no expirados) fácilmente y que se podía trabajar desde el primer momento con un caso real, la solución ha sido utilizar el certificado personal, lo que conlleva prestar especial atención al actualizar el proyecto en el control de versiones remoto (GitHub) para que no quedaran rastros de información personal.

Una vez resueltos estos problemas, se accede a la página de bienvenida de Web-eID (ruta */welcome*), donde se muestra el fichero de ejemplo y el botón para comenzar el proceso de firma de dicho fichero, así como un botón para cerrar sesión. Es interesante mencionar que si se intenta acceder a esta ruta directamente sin iniciar sesión o si pasa un tiempo y se recarga la página, se obtiene un error 403 (no autorizado). Esto indica que se mantiene una sesión que expira al cumplir la fecha de expiración del token jwt enviado desde la aplicación nativa.

[27] https://github.com/web-eid/web-eid-authtoken-validation-java

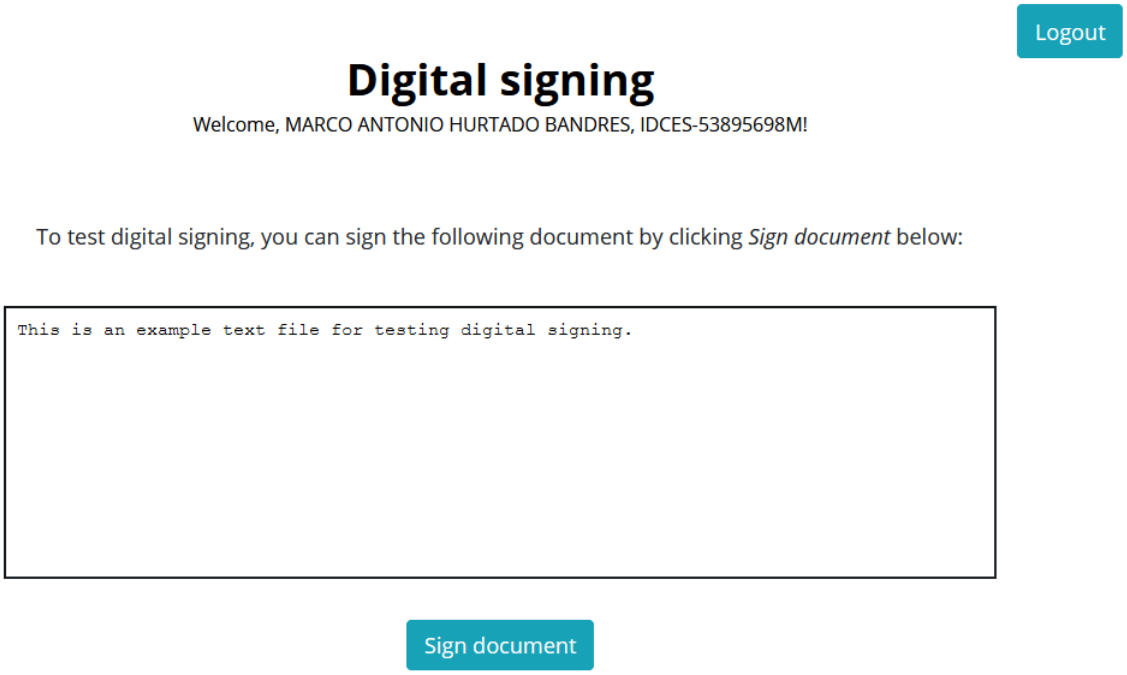

Figura 10: Página de bienvenida de Web-eID.

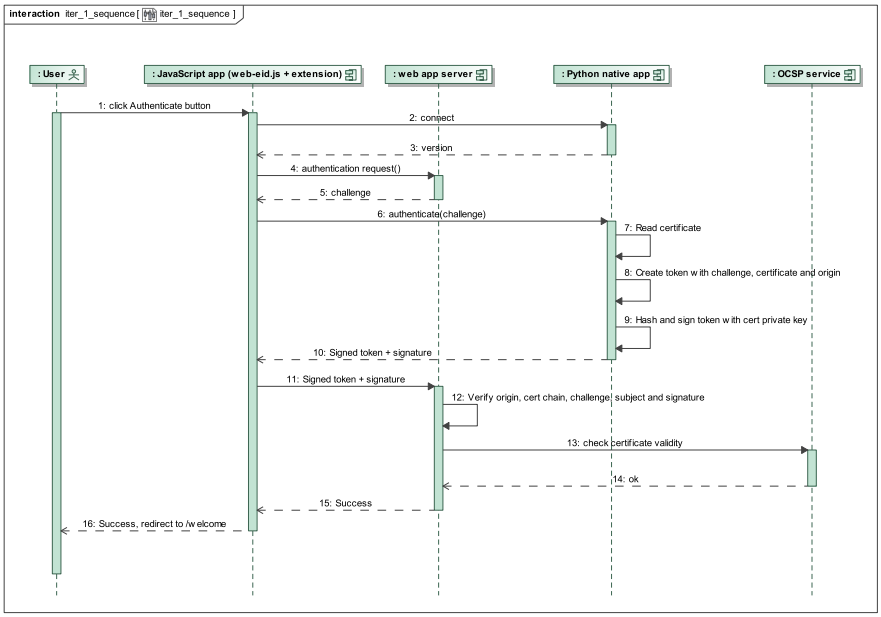

Figura 11: Diagrama de secuencia de la fase de autenticación.

6.4. Iteración 2. Firma

6.4.1. Iteración 2.1. Get-Certificate

Como se puede comprobar en el diagrama de flujo del proceso de firma de Web-eID 5, hay un paso previo a la realización de la firma reflejado en el comando *get-certificate*, que consiste en **tomar el el certificado de clave pública de la tarjeta para pasarlo al servidor de la aplicación web** y que éste cree el contenedor de firma (en formato *asice*) con el documento y calcule el hash, que será posteriormente enviado a la aplicación nativa para que, esta vez sí, sea firmado con la clave privada de la tarjeta.

En esta iteración se reproducirá este paso al igual que en el proceso de autenticación, con el certificado en un fichero en lugar de la tarjeta. Como realmente esta funcionalidad ya se había cubierto en el paso de la autenticación, pues era necesario leer el certificado para crear el token de autenticación, solamente se ha extraído dicha funcionalidad en el manejador *readCertificate* para poder reutilizarla en autenticación y en firma. Además, se han añadido algunas comprobaciones de errores a la hora de recibir los comandos tanto de autenticación como de este primer paso de firma.

En esta fase no se han encontrado problemas mayores, y ha sido casi principalmente un proceso de traducción de la aplicación nativa de C++. No obstante, la funcionalidad se ha limitado al tipo de certificado de la AC-FNMT Usuarios, con algoritmo de hash SHA-256 y firma SHA256 con RSA. El soporte para más algoritmos de firma y de hash se ha dejado como característica de menor prioridad para iteraciones futuras.

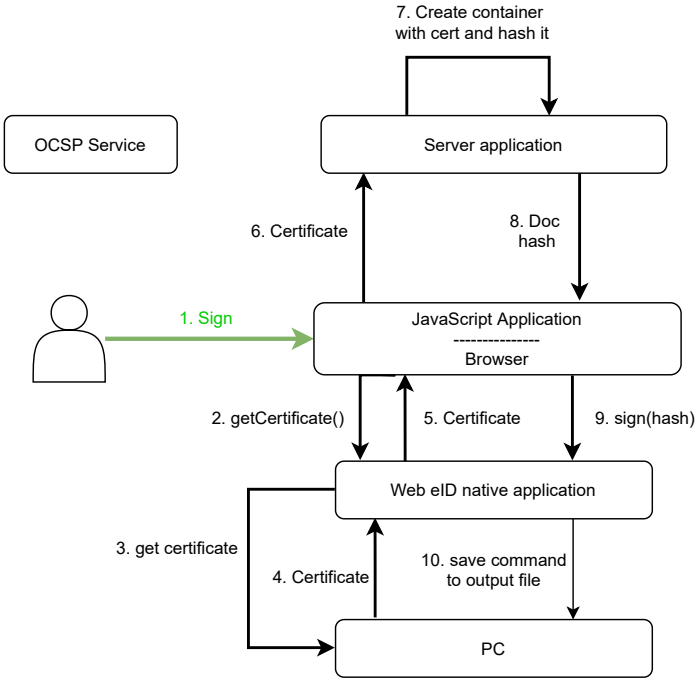

Figura 12: Diagrama de flujo de la primera fase de firma.

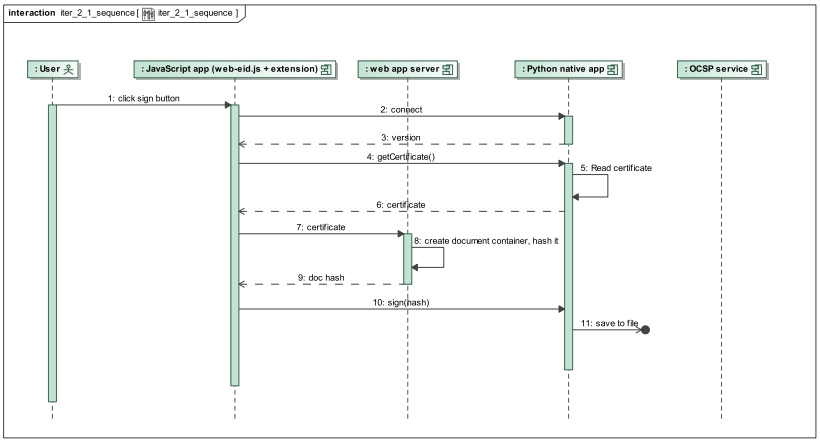

Figura 13: Diagrama de secuencia de la primera fase de firma.

6.4.2. Iteración 2.2. Firma

Esta iteración ya cubrirá el proceso de firma propiamente dicho. **Lo que se firmará** en la aplicación nativa con la clave privada del fichero del usuario **es el resultado de aplicar una función de hash al contenedor de firma**, de forma que el servidor de la aplicación web, al recibir la firma, la valide y la añada al contenedor. Además, previa confirmación de que la firma se ha realizado correctamente, es necesario verificar el estado del certificado de firma con un servidor OCSP.

Este proceso también ha sido principalmente de traducción, realizando la acción de firma con funciones de la librería Cryptography de Python. Además, **como se firma un hash, es necesario indicar que los datos a firmar ya han pasado por una función de hash** previo. Para ello se pasa una instancia de *Prehashed*[28].

Por parte del servidor de la aplicación, el contenedor creado satisface el estándar **ASiC**[29] y la firma que se realiza es **XAdES** (XML Advanced Electronic Signature). La firma XAdES concreta una serie de perfiles para que la firma sea reconocida, y particularmente, tal y como está implementado el ejemplo de Web-eID, se realiza una firma con perfil XAdES-T, es decir, **con un sello de tiempo** que garantiza el no repudio. Los distintos perfiles esencialmente añaden elementos a la estructura básica definida por el estándar XAdES del archivo XML, y dicho archivo se incluye dentro de la carpeta *META-INF* del contenedor [47].

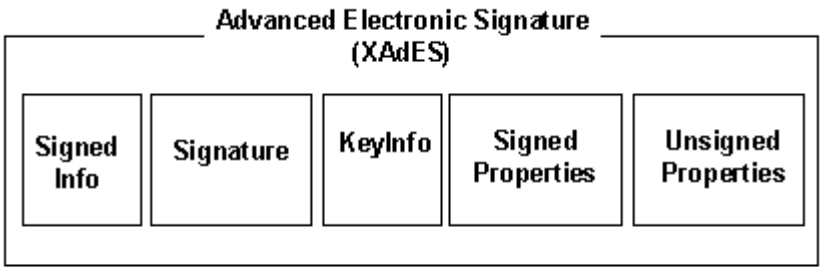

Figura 14: Elementos de la firma XAdES [47]

[28] Método sign de cryptography

[29] https://www.cryptomathic.com/news-events/blog/asic-associated-signature-containers-for-eidas

Los **problemas** en esta fase han sido los que han requerido más tiempo para solucionar y se han producido al validar e incorporar la firma al contenedor. A continuación, se explican sin entrar en demasiados detalles técnicos cómo se intentaron resolver.

- **Problema: no es posible finalizar la firma del contenedor ni validarla al no contener respuesta del OCSP.** Esto provocaba que al comprobar los campos de la firma y ver que el de la respuesta del OCSP esataba a nulo, no se pudiera validar ni finalizar el contenedor correctamente.

- **Primer intento. Actualización de la librería**, pues el proyecto de la aplicación web de ejemplo había tenido una nueva *release* que solo actualizaba la versión de Digidoc4j. Los principales cambios de la nueva versión (4.1.1) hacen referencia principalmente a una actualización de listas de confianza, que podía resolver potencialmente fallos al realizar o validar firmas[30], y se pensó que podía resolver el problema, pero no fue así. Si bien es cierto que la librería ha recibido otra actualización, este hecho se ha conocido demasiado tarde como para arriesgarse a hacer cambios mayores en la prueba de concepto.

- **Segundo intento. Ejecución de tests de Digidoc4j:** para comprobar que la librería funcionase correctamente de forma aislada, se optó por clonarla y ejecutar sus tests. Los resultados de estos tests fueron fallos en muchos casos, fallos que podrían deberse a la configuración del entorno de Java, pero antes de entrar en esos ajustes, se hizo otro intento.

- **Tercer intento. Extraer y probar los tests de Digidoc4j relevantes:** en lugar de ejecutar la batería de tests, se analizaron cuáles probaban las funciones que interesaban, se extrajeron en un proyecto por separado con la librería en su totalidad en lugar de por dependencias de maven y se ejecutaron tanto con certificados españoles como de Estonia, pues en el proyecto se encontraron algunos. El resultado no fue exitoso, pero aislar el problema permitió una depuración más profunda.

Si bien es cierto que la finalidad principal de este TFG no era replicar fielmente todos los procesos de autenticación y firma, se ha realizado un último intento basado en Autofirma, es decir, **reemplazar el proceso de firma con las herramientas de autofirma.**

[30] https://github.com/open-eid/digidoc4j/releases/tag/public-4.1.1

6.4.3. Iteración 2.3. Intento de incorporación de Autofirma y solución final

En esta iteración, el objetivo ha sido **hacer un análisis de los procesos de firma de Autofirma que son aplicables al caso de estudio**, con el objetivo de compararla con Digidoc4j y analizar si se podría integrar como sustituto de esta librería en la prueba de concepto.

Para ello, se ha procedido de forma similar al caso de Digidoc4j: **extraer los tests** con las funcionalidades que interesaban, **replicarlos** con el certificado personal (era el que se seguía empleando), **depurar** los pasos del proceso con las llamadas entre clases y parámetros que se utilizan y **probar la integración con la aplicación de web-eID** mediante un nuevo *SigningService*. Este servicio utilizaría los métodos y clases de Autofirma para crear el contenedor y añadirle posteriormente la firma recibida de la aplicación nativa.

El resultado de este análisis ha sido que, en primer lugar, **se ha encontrado un certificado de prueba no expirado**, lo cual ya permitiría incluso empaquetar el proyecto de la prueba de concepto con un certificado de prueba funcional y válido.

Entrando en aspectos más técnicos, se ha comprobado que **las clases *signers*** de Autofirma (las clases que realizan la firma) **requieren una clave privada por parámetro para poder firmar**. Además, la construcción del contenedor de firma se realiza dentro del proceso de firma, no habiendo ningún caso como el de Web-eID en el que primero se cree el contenedor y luego se complete añadiéndole la firma. Esto tiene sentido ya que el caso de uso de Autofirma es el usuario firmando con su clave directamente en la aplicación. Por tanto, no está preparada para iniciar un proceso de firma y que un paso se realice en un componente diferente. De hecho, al crearse el contendor directamente con la firma en Autofirma[31].

Este hecho por tanto, sugiere que **de integrar Autofirma en la estructura de la solución, se situaría en el lado de la aplicación nativa** y que en un futuro, se puede contemplar el caso de lanzar Autofirma desde la aplicación de Python, dando las opciones de realizar la firma tanto en la aplicación nativa como en Autofirma.

Finalmente, la **solución final preventiva** por la que se optó no resolvía el problema en su totalidad. Primero, se ha añadido la librería en su totalidad en lugar de a través de dependencias y luego, se ha eliminado la comprobación que causaba el error y se ha hecho por otro medio.

[31] Generador de contenedores ASiC de Autofirma

Ésta comprobación es una petición OCSP realizada tras añadir la firma al contenedor, que toma la dirección del OSCP directamente del certificado del usuario. Para ello, se ha reutilizado y extraído a un servicio parte de la implementación de la librería de validación de tokens de Web-eID.

No obstante, y en iteraciones posteriores, **se ha conseguido resolver el problema cambiando el perfil de la firma XAdES**. Como se mencionó, se hacía una firma XAdES-T, con sellado de tiempo, pero al comparar la firma que realiza Web-eID y la de Autofirma con una herramienta web[32], se comprobó que **Autofirma realiza una firma XAdES-B**, que según la especificación incluye menos elementos a validar [47]. **Sustituyendo el perfil de firma a XAdES-B, ya ha sido posible trabajar con la librería Digidoc4j sin alteraciones**. El problema se ha considerado resuelto al obtener el mismo resultado en la validación de los contenedores generados por Autofirma y por el ejemplo modificado de Web-eID con la herramienta web mencionada, pero es importante señalar que al validar la firma tanto en Autofirma como en la aplicación de escritorio de Digidoc, se obtiene una advertencia indicando que no se ha podido realizar la validación completa de la firma.

En lo que respecta a la **aplicación nativa de Python**, otras funcionalidades añadidas en este período tienen que ver con validaciones adicionales y clases con funciones útiles, como la de convertir un algoritmo de firma a una tupla *{algoritmo criptográfico, algoritmo de hash, algoritmo de padding }* tal y como se hace en la aplicación nativa original.

Esta iteración ha servido para obtener una mayor información sobre la implementación de firmas avanzadas (XAdES, CAdES, etc.) y para tener una perspectiva de la complejidad que supone implementarlas en un lenguaje desde cero, siendo indispensable utilizar librerías externas.

Una vez acabada la modificación de la aplicación web de ejemplo, **se ha alojado en heroku** para que sea accesible sin necesidad de levantar el proyecto[33].

[32] https://dss.nowina.lu/validation
[33] https://webeidspringexamplemodified.herokuapp.com/

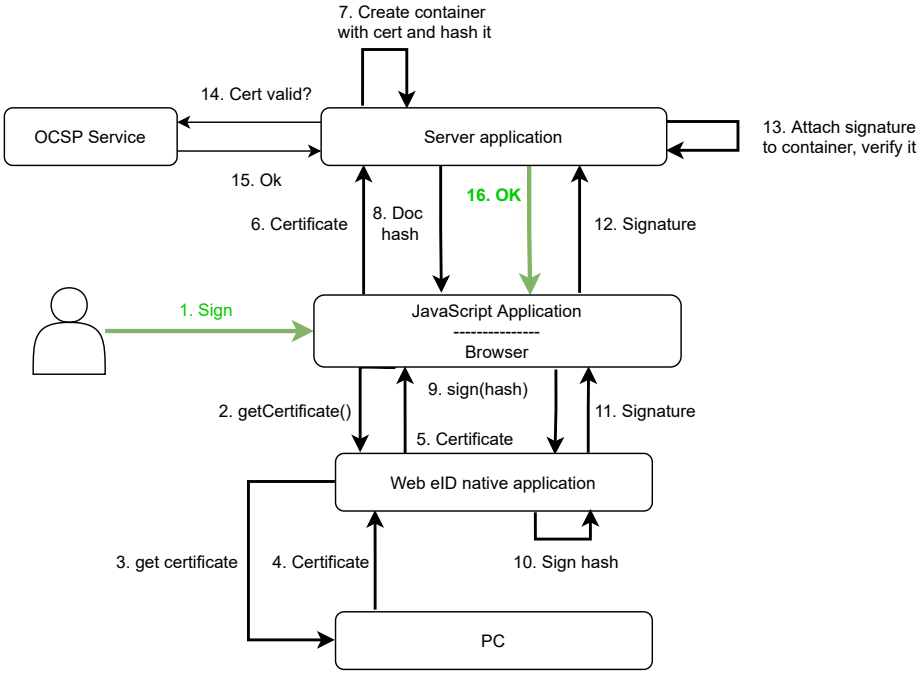

Figura 15: Diagrama de flujo del proceso de firma modificado.

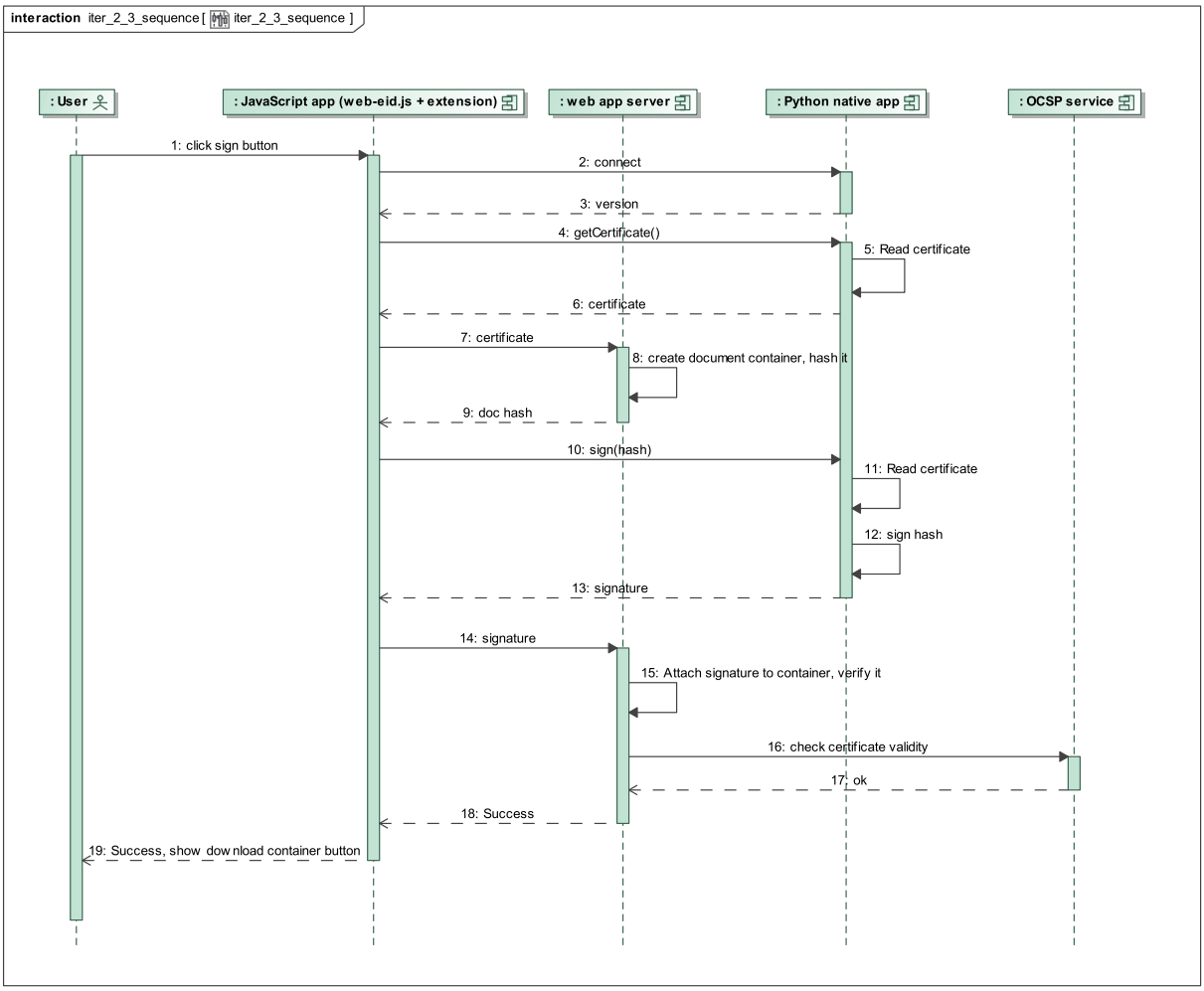

Figura 16: Diagrama de secuencia del proceso de firma modificado.

7

Fase 2. Implementación de la solución de delegación de firma

7.1. Iteración 1. Configuración y pruebas iniciales del servidor de registros

Esta fase es la dedicada al propósito principal de la prueba de concepto: **implementar los mecanismos de control de delegación de firma**. Desde la investigación previa se ha mencionado tanto los registros de uso del certificado en un servidor remoto consultable por el usuario, como los permisos de uso. En esta primera iteración se utilizarán **entidades de prueba** de registros (es decir, sin los campos de datos definitivos), y se irán añadiendo medidas de seguridad. De esta forma, al determinar los campos definitivos de los registros, será una adaptación casi inmediata, añadiendo elementos en lugar de modificar los existentes.

Para este fin se ha optado por **utilizar como base la aplicación web modificada en la fase previa**, reutilizando la autenticación con el certificado como medida de acceso a la consulta de registros del mismo y eliminando o inhabilitando la función de firma. A esta estructura, se le añadirá una conexión a una **base de datos remota** (MongoDB Atlas), puntos de acceso (*endpoints*) al servidor para recuperar, modificar, crear, etc. los registros almacenados, y **se modificarán algunos componentes** de la estructura para que se comuniquen con el servidor de registros.

El **flujo de registros de uso en autenticación y firma será similar**, consistiendo en en insertar los registros en la base de datos vía petición POST desde la aplicación nativa para reflejar que se ha realizado un intento de autenticación, y desde la extensión modificar el estado de dicho registro si ha tenido éxito la autenticación o firma. Además, lo ideal es enviar un correo electrónico a la dirección que se suele encontrar en el campo *subject alternative names* del certificado (pero puede no tenerlo), para informar al titular del certificado. También se guardará un registro en un archivo local para no depender de Internet en todo momento. Los flujos y secuencias de autenticación y firma quedarían de la siguiente forma.

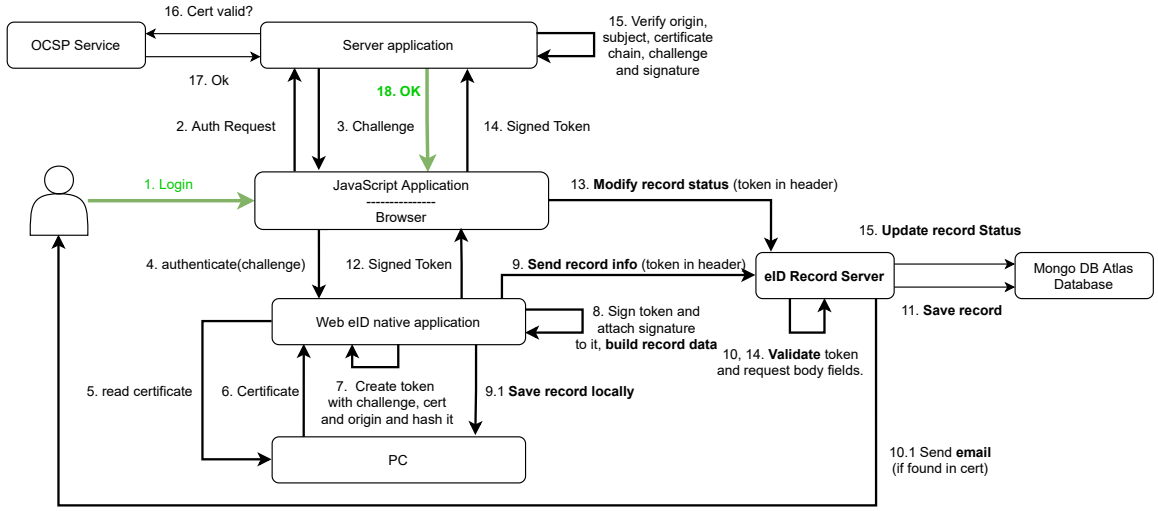

Figura 17: Diagrama de flujo del proceso de autenticación con registros de uso del certificado.

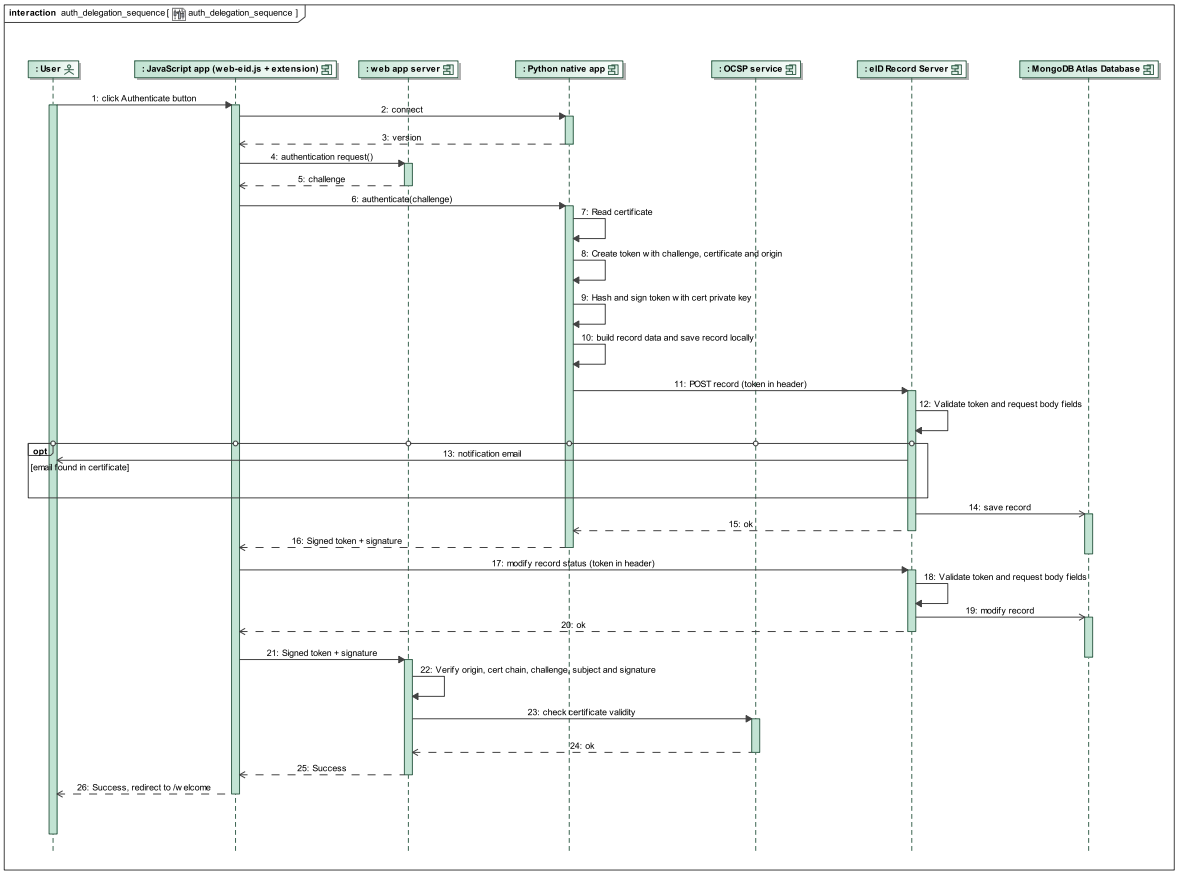

Figura 18: Diagrama de secuencia del proceso de autenticación con registros de uso del certificado.

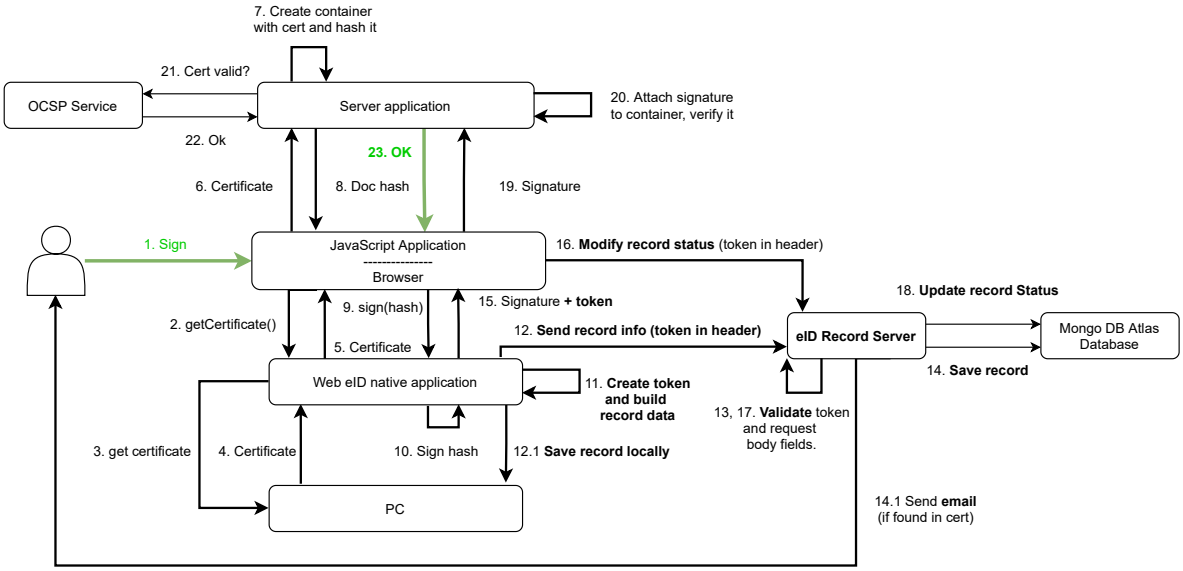

Figura 19: Diagrama de flujo del proceso de firma con registros de uso del certificado.

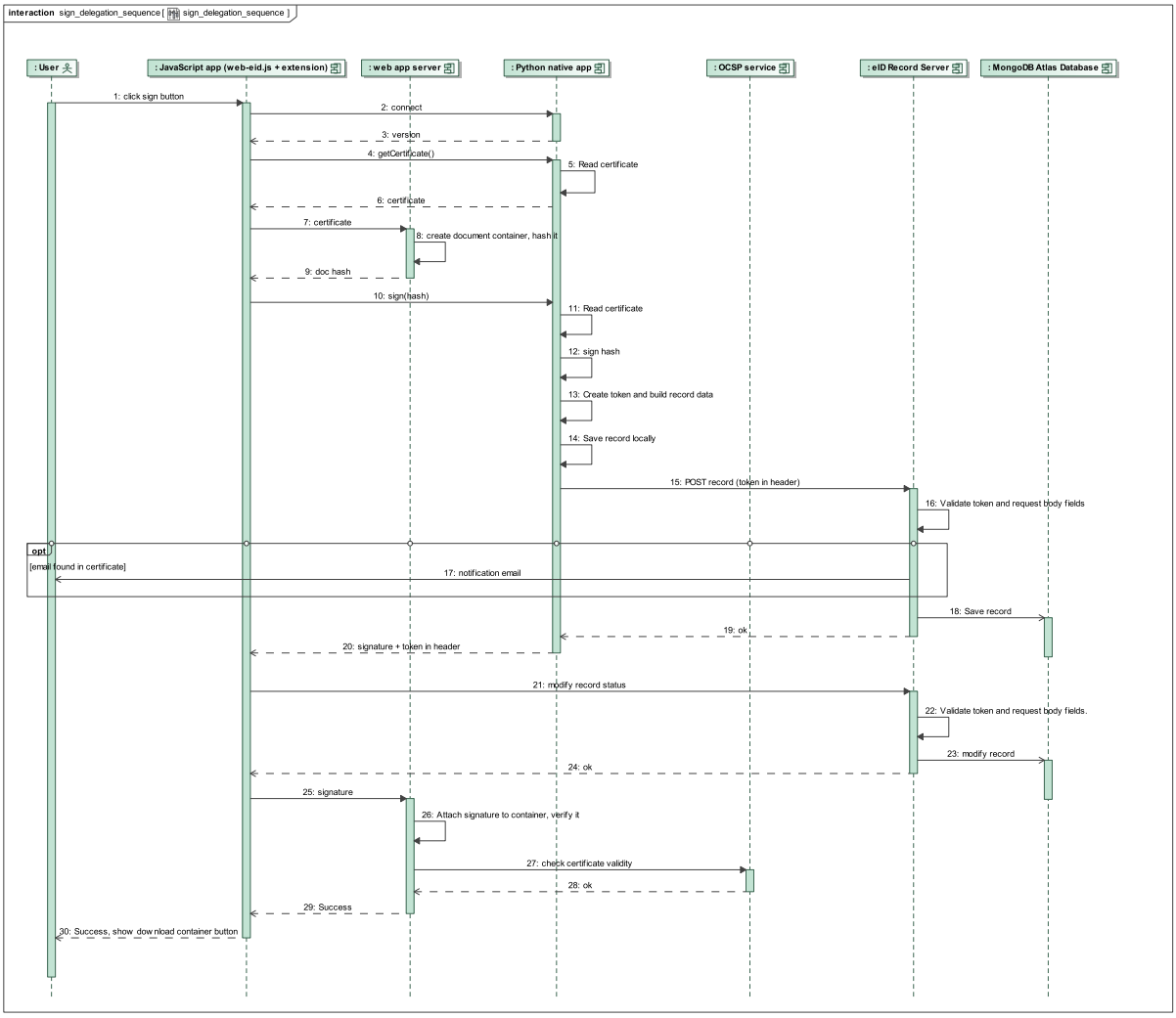

Figura 20: Diagrama de secuencia del proceso de firma con registros de uso del certificado.

En cuanto a la **seguridad de la base de datos**, es importante un diseño correcto con el objetivo de evitar inserciones de registros con cualquier contenido. Esto es especialmente importante en este caso al utilizar una base de datos NoSQL, pues los documentos pueden no tener determinados campos y ser almacenados igualmente. Por tanto, la protección de la base de datos no ha pasado tanto por medidas de cifrado de la base de datos, sino por la propia **protección del servidor, al ser éste el único que se comunica directamente con ella**. Las medidas tomadas han sido:

- **Obligar a incluir un token de autenticación en las cabeceras de las peticiones.** En el caso del proceso de autenticación, se ha utilizado el **token jwt** que se envía a

la extensión. De esta forma, la aplicación nativa puede autenticar sus peticiones POST y la extensión, sus peticiones PUT (de modificación). En el caso del proceso de firma, sí que se ha tenido que **construir el token desde cero**, pero no ha sido complejo, al disponer de toda la información necesaria para construirlo. Lo único que se ha modificado es el campo del *nonce*, que se ha sustituido por el hash del contenedor, al haberse comprobado que es indeterminista (no se obtiene la misma salida con la misma entrada). También, al tener que utilizarlo la extensión, se ha añadido el campo *token* a los datos recibidos por ella desde la aplicación nativa. Sobre este token, se han realizado las siguientes comprobaciones:

  - **Comprobar que el token está** en la cabecera.
  - Verificar que **posee un *nonce*** (el hash del contenedor en el caso de la firma) y que **es igual al del cuerpo de la petición** (que contiene la información del registro).
  - **Validar el estado del certificado** del token.
  - **Validar la firma del token.**

- **Incluir una entrada por cada certificado del que haya registros.** Esta medida va enfocada a garantizar que **solo se almacenan certificados en buen estado** y además, al editar un registro, antes se comprueba que el certificado al que corresponde se encuentra en la base de datos, lanzando un error en caso contrario.

Es importante mencionar que no se puede utilizar el mecanismo completo de validación de tokens de la librería *web-eid-authtoken-validation-java* en la validación de los tokens de las cabeceras de las peticiones recibidas debido a que esta validación se basa también en un cacheo de tokens y de nonces con el objetivo de mantener una sesión autenticada en la aplicación web.

Finalmente, el **envío de correos electrónicos** dada la estructura de la solución, se puede realizar desde la aplicación nativa o desde el propio servidor de registros. La ventaja de hacerlo en la **aplicación nativa** es que en Python es más sencillo, pero hay un **riesgo de seguridad importante**, pues hay que almacenar el usuario y la contraseña de la cuenta de correo del servidor de registros de forma segura. Un acercamiento posible es el almacenarlos como variables de entorno en un archivo *.env*. No obstante, de cara a la distribución de la aplicación nativa, implicaría incluir el archivo, exponiendo las credenciales.

Por tanto, **la opción más segura es hacerlo desde el servidor de registro**, pues al alojarlo en heroku, las variables de entorno se almacenan en la nube, siendo inaccesibles por terceros. Sin embargo, a la hora de levantar el proyecto del servidor en local, se debe desactivar la opción de envío de correos al no contar con las credenciales directamente en el proyecto.

7.2. Iteración 2. Inicio de la versión definitiva del servidor de registros

Esta iteración se ha centrado en **comenzar a desarrollar la versión definitiva del servidor**. Para ello, se han creado en primer lugar las **colecciones y clases de registros** tanto en el servidor como en la base de datos. De esta forma, la información que se almacena en el registro (clase *EidRecord*) es:

- **_id**: identificador que posee todo documento almacenado en una base de datos de MongoDB.
- **Fecha** del registro.
- **Plataforma:** tipo de máquina desde la que se llevó a cabo la actividad.
- **IP pública** de la máquina.
- **Acción:** autenticación o firma.
- **Origen:** URL desde la que se realizó la autenticación o firma.
- **Nonce:** este es el **identificador único del registro** al ser una cadena aleatoria. como se ha indicado, en el caso de la autenticación es el nonce generado por el servidor, y en el caso de la firma, el hash del contendor.
- **Email** del sujeto del certificado, si lo posee.
- **Huella del certificado:** resultado de aplicar un hash de tipo SHA256 a los bytes del certificado. Sirve para recuperar el certificado a partir del registro, pues es el **identificador único de la entidad del certificado.**
- **Estado** de la acción. Será *Python native app success* y luego, si se completa con éxito, será modificado por la extensión.

Además, **los certificados se almacenan envueltos en la entidad *EidPublicCertificate*** con la siguiente información:

- **_id** de MongoDB.
- **Bytes del certificado**.
- **Nombre distinguido del sujeto.**
- **Nombre distinguido del emisor.**
- **Email** del sujeto del certificado, si lo posee.
- **Huella del certificado:** como se ha mencionado, es el SHA256 de los bytes del certificado y sirve como **identificador único de la entidad.**

7.3. Iteración 3. Modificación del flujo de firma y modificaciones finales

En esta iteración final, se añadió una comprobación adicional, y es **validar la firma cuando el servidor recibe un registro de firma.** Esto implicaba volver a modificar lo que envía la aplicación nativa a la extensión tras firmar, añadiendo la cadena en base64 de la firma. Esto también se incluirá en una **cabecera adicional** en las peticiones al servidor de registros, de forma que el este, al recibir una petición para registrar o modificar un evento de firma, tome la firma de la cabecera y la valide.

El principal **problema** que conllevó es que **en Java no se conseguía validar la firma de Python**, pues recuérdese que en Python se utilizaba una instancia de *Prehashed*. La **solución preventiva final**, surgida tras hacer varias comparaciones de las formas de firma en ambos lenguajes, fue **realizar una firma con *Prehashed* y otra sin *Prehashed*** en la aplicación nativa, ya que en Java se conseguía validar la segunda, por lo que desde Python se utilizaría la firma sin *Prehashed* para la petición al servidor de registros y pasaría ambas firmas a la extensión, para que esta envíe la primera a la aplicación donde se realiza la firma, y la segunda en una cabecera con la petición de modificación al servidor de registros.

No obstante, esto suponía un aumento de la cantidad de información en tránsito, y tiempos adicionales. Además, en las comparaciones de la firma, se vio que firmando el contenedor entero (que además no es mucha cantidad de información, sino más bien metadatos), ya no era necesario hacer uso de la instancia de *Prehashed* y tanto el servidor de registros como el servidor de la aplicación web de autenticación y firma la aceptaban. Por tanto, **se modificó el proceso de firma**, manteniendo el flujo, pero **enviando y firmando el contenedor entero en lugar del hash de este.** Esta modificación, además, **permite almacenar la información del contenedor firmado en local**, teniendo más detalles de qué se firma.

Tras finalizar esta modificación, **se han añadido tres funcionalidades menores**.

1. **Activar y desactivar notificaciones por correo electrónico** desde la vista de la lista de registros. Esto requirió añadir una propiedad a la entidad *EidPublicCertificate* para indicar si para ese certificado se deben enviar o no correos electrónicos de notificación.

2. **Posibilidad de descargar registros en formato JSON** desde la vista de la lista de registros, recuperándolos desde la base de datos con el identificador (*tokenNonce*).

3. **Posibilidad de descargar un *mock* del fichero firmado** desde la vista de la lista de registros. Esto requirió añadir una propiedad a la entidad *EidRecord* (*fileId*) consistente en un identificador hexadecimal aleatorio generado en la aplicación nativa solamente para los registros de firma. **Lo que se descarga el usuario siempre es el mismo fichero por defecto** (un *mock*) independientemente del registro, pero se le ha asociado este identificador. Realmente esta funcionalidad es un comienzo para una posibilidad de futuras versiones de la solución. Además, **también se almacena la cadena en base64** de los datos firmados, para así poder tenerlos en local (almacenados directamente al recibirlos de la extensión) y en remoto (convirtiendo a fichero dicha cadena).

El flujo de firma final, por tanto, queda como sigue:

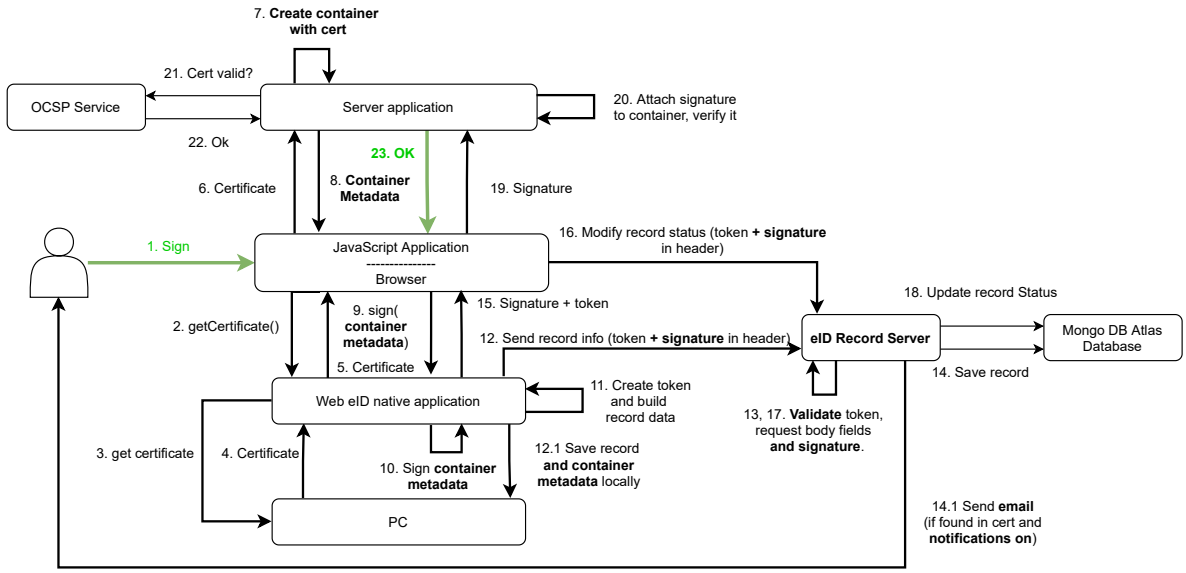

Figura 21: Diagrama de flujo final del proceso de firma con registros de uso del certificado.

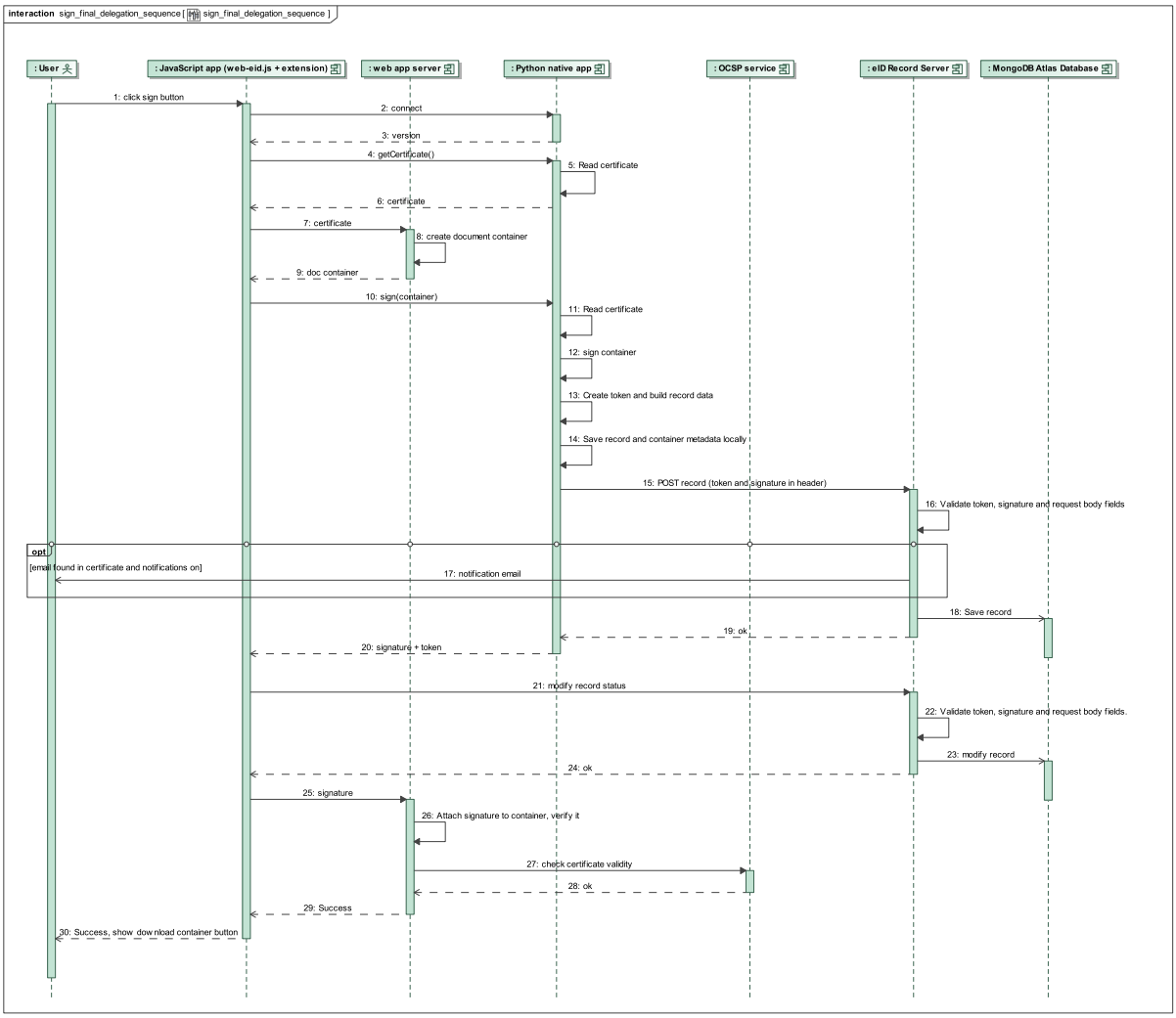

Figura 22: Diagrama de secuencia final del proceso de firma con registros de uso del certificado.

8

Conclusiones y Líneas Futuras

8.1. Conclusiones

En primer lugar, y haciendo referencia a la **legislación**, si bien es cierto que se persiguen objetivos muy positivos como los de interoperabilidad y la agilización de trámites, **no se menciona prácticamente nada sobre la delegación de firma, a pesar de que es un mecanismo que precisamente agiliza enormemente los trámites.** A nivel de España, las cosas tampoco cambian, pues **en ninguno de los Boletines Oficiales del Estado referenciados se menciona la delegación de firma** ni en qué afecta eIDAS a ello. Por lo general, ha sido complejo encontrar artículos y documentación que relacionen las nuevas normas europeas de identidad digital y la delegación.

En cuanto a las soluciones analizadas, se ha demostrado tanto a nivel teórico como práctico que **sean soluciones del sector público o privado e independientemente de las tecnologías empleadas, se pueden adaptar para cubrir con más seguridad la delegación**. Simplemente añadiendo historiales tanto locales como remotos se daría un paso de gigante, y si muchas soluciones que no se basan en la nube adaptasen medidas de estas, el escenario general sería mucho más cercano al ideal. A día de hoy **las soluciones cloud son las que están más preparadas para garantizar una delegación de firma segura**, pero con el desarrollo de la prueba de concepto se ha puesto de relieve no solo que es posible añadirlo a una solución basada en extensiones y ya funcional en algunos Estados europeos, sino que potencialmente es extensible esta solución al hardware, pudiendo en un futuro disponer de una solución europea polivalente e integral. Esto es precisamente lo que persiguen algunas de las proposiciones mencionadas en el apartado de legislación, como la Identidad Digital Europea propuesta en el

ámbito de la Brújula Digital de la Unión Europea para 2030, y que necesita una arquitectura debajo que pueda ser utilizada por todos los países de la unión [18].

En España se aprecia la **adaptación paulatina al reglamento eIDAS**, pero aún tiene un camino por recorrer importante, no solo por la necesidad de desarrollar o mejorar los sistemas de identidad digital públicos, sino también por la necesidad de eliminar todo lo obsoleto en pos de una solución estándar en forma de aplicación o de estándar tecnológico que utilicen todas las Administraciones Electrónicas.

Tal y como se ha señalado previamente, las soluciones públicas analizadas (Cl@ve y @firma) pueden hacer la delegación más segura con las medidas propuestas tanto en el ámbito del uso por el ciudadano en su día a día (Cl@ve) como en las administraciones (@firma).

Los **requisitos** principales que se han obtenido del la investigación previa son comunes a todas las soluciones, habiendo algunos concretos según la tecnología (los mecanismos de anulación del hardware por ejemplo), que es **como mínimo registrar lo que se realiza con el elemento que se delega en todo momento y que luego el titular de este pueda consultar el historial de uso**. Esto probablemente se realice de alguna forma que no se explica en muchas de las soluciones analizadas, pero lo que no hacen muchas es poner este historial de registro a disposición del usuario. Por tanto, para mejorar la confianza de este y fomentar el uso de los medios electrónicos, **debe ofrecerse transparencia total**. Otras medidas adicionales como los **permisos de uso** quizás son más útiles en el ámbito administrativo.

Como conclusiones del **desarrollo de la prueba de concepto**, en primer lugar, decir que **la solución de Web-eID va en una dirección interesante** y adecuada para cumplir los objetivos de interoperabilidad de eIDAS aunque no se centre en la delegación de firma, pero como se ha comprobado, se puede modificar un componente para una mayor flexibilidad de documentos de identidad manteniendo el resto de componentes poco alterados, y no solo eso, sino también añadir el soporte simplemente con el servidor de registros y los registros locales. Esto indica que **la estructura de la solución en sí está bien diseñada** al ser expansible y con componentes reemplazables sin dificultades extremas.

En segundo lugar, y a partir de los problemas encontrados, es muy importante **mantener la documentación lo más completa y actualizada posible**, pues **ahorra mucho tiempo**

por un lado, de intentos fallidos evitables con una simple indicación o advertencia de que una solución no está soportada en un determinado entorno, y por otro, de *buceo* en un océano de documentación sin orden y con referencias a componentes ya obsoletos (caso de @firma y de Vsmartcard). Finalmente, y relacionado con la **transparencia y la documentación**, se deben **proporcionar y mantener actualizados los certificados de prueba**. De esta forma se fomenta que desarrolladores independientes investiguen formas de mejora de las aplicaciones públicas. En el caso de España el problema ha sido no mantener actualizado el set de certificados de prueba del DNIe, pero el caso de Web-eID es más grave al requerir tarjetas de prueba (de pago) [34] y tener que subir los certificados para utilizar los OCSPs de prueba[35]

En definitiva, para **mejorar la experiencia de usuario de delegación de firma en la administración electrónica pública**, se debe, en primer lugar, **simplificar el funcionamiento de las soluciones de identidad digital estatales**. Esto implica que de cara al usuario deben ser sencillas tanto de emplear como de comprender, sin que tenga que conocer demasiadas aplicaciones debido a que o bien cada Administración Electrónica tenga una, o bien que para cada tipo de trámite haya una aplicación distinta. En segundo lugar, **estas deben ser robustas**, pues se emplean para realizar muchísimos trámites diarios, con datos sensibles y con riesgos importantes. Y finalmente, **otorgar el mayor nivel de seguridad posible al usuario**. Esto implica varios factores como la **transparencia** y **seguir principios de diseño de software en todos los aspectos**, desde la interfaz de usuario hasta el código. En este TFG se ha propuesto una posible solución basada en **registrar el uso de un certificado**, **notificar** por correo electrónico (si se tienen las notificaciones activadas) y dar la posibilidad de **acceder y descargar los registros**, pero como se verá en el siguiente apartado no es la única opción.

El primer paso, no obstante, es la **concienciación de la población** sobre la importancia de proteger nuestra identidad tanto física como digital. El conjunto de la sociedad debe comprender que **tan importante es proteger tus certificados digitales como tu documento de identidad físico**.

[34] https://www.id.ee/en/article/service-testing/

[35] https://www.id.ee/en/article/service-testing/#other-test-services

8.2. Líneas Futuras

Este trabajo no tenía el principal objetivo de desarrollar una prueba de concepto, y por ello se ha centrado en gran medida en hacer una investigación previa importante, para servir como una base sobre la que se pueda construir y refinar una solución o soluciones que blinden lo máximo posible la delegación de identidad digital. Obviamente y como es común en el desarrollo de software, los requisitos obtenidos no son definitivos, y en un futuro se pueden añadir o completar.

En cuanto a la prueba de concepto desarrollada, ha empleado la arquitectura de Web-eID, demostrando que es extensible y con componentes reemplazables. Muchas líneas futuras pueden pasar por aspectos como **modificar más componentes o extender la arquitectura.** Otras opciones pueden completar algunos de los intentos de diseño iniciales e integrarlos a esta, como el uso del dispositivo de Nordic, e incorporarlo a la estructura o diseñar una nueva.

Otra posibilidad de extensión e integración tiene que ver con **soluciones públicas como Autofirma**, pues se intentó incorporar parte de ella, pero cabe la posibilidad de **incorporarla completamente**, por ejemplo lanzándola desde la extensión, siendo una alternativa a la aplicación nativa de Python, o desde la propia aplicación nativa, haciendo esta de proxy con otras aplicaciones de escritorio potencialmente. Finalmente, también es interesante investigar cómo realizar una versión de Valide, Autoscript y Autofirma con soporte seguro para delegación de firma. Como se indicó en el punto de mejoras de las soluciones públicas, se podría hacer uso de una extensión, adaptando estructura de Web-eID a las aplicaciones de las que ya disponemos en España.

Entrando en **mejoras de lo desarrollado**, la aplicación nativa tiene el punto débil de **almacenar la contraseña del certificado en plano en un archivo de entorno**. Esto imposibilita su uso en un escenario real, por lo que sería necesario encontrar la forma de almacenarlas y recuperarlas de forma segura, para acercar la solución integral progresivamente a un *producto final.* Además, ya que se ha visto que la arquitectura de Web-eID se puede integrar con certificados españoles, es interesante **investigar la forma de incluir el uso de estos en la aplicación nativa original**, pudiendo emplear en consecuencia tanto certificados como tarjetas inteligentes (es decir, hacer un **fork** de la aplicación nativa). Otras mejoras estéticas

pasan por incluir **interfaces de usuario** que den feedback visual al usuario en lugar de los logs actuales únicamente.

Más aspectos mejorables pueden ser la **seguridad de la base de datos**, pues podría cifrarse para garantizar una mayor protección, aunque en esta prueba es el servidor el único componente con acceso directo a ella. Otra funcionalidad para acercar la solución a un producto más real es **subir los archivos que se quieran firmar**. De hecho, la aplicación de ejemplo de Web-eID tiene esta funcionalidad en código pero comentada[36], por lo que se podría explorar. Finalmente, se puede buscar la forma de **almacenar lo que se firma** para no descargar un *mock* desde el servidor de registros, sino el archivo real firmado. Si es demasiado grande, quizás se puede ofrecer en forma de metadatos, o comprimida.

Sin duda, estos han sido los primeros pasos a nivel de investigación y de desarrollo para mejorar la seguridad del proceso de delegación de identidad digital, proceso que es cada vez más común y frecuente en el día a día de todos los ciudadanos, y que por tanto es cada vez más necesario proteger.

[36] Servicio de firma de Web-eID con código para subir archivos comentado.

Bibliografía

[1] The Web eID project. *"web-eid/web-eid-authtoken-validation-java: Web eID authentication token validation library for Java", github.com.* URL: https://github.com/web-eid/web-eid-authtoken-validation-java (visitado 17-06-2021).

[2] Open eID. *"open-eid/digidoc4j: DigiDoc for Java", github.com.* URL: https://github.com/open-eid/digidoc4j (visitado 25-06-2021).

[3] Open eID. *"JavaMail", javaee.github.io.* URL: https://javaee.github.io/javamail (visitado 25-06-2021).

[4] Connect2id. *"JOSE + JWT library for Java | Connect2id", connect2id.com.* URL: https://connect2id.com/products/nimbus-jose-jwt (visitado 25-06-2021).

[5] Spring. *"Spring Boot", spring.io.* URL: https://spring.io/projects/spring-boot (visitado 25-06-2021).

[6] Microsoft. *"IDE de Visual Studio, editor de código", visualstudio.microsoft.com.* URL: https://visualstudio.microsoft.com/es/ (visitado 25-06-2021).

[7] Segger. *"Embedded Studio: TheCross-Platform IDE", segger.com.* URL: https://www.segger.com/products/development-tools/embedded-studio/ (visitado 25-06-2021).

[8] Microsoft. *"Visual Studio Code - Code Editing. Redefined", code.visualstudio.com.* URL: https://code.visualstudio.com (visitado 25-06-2021).

[9] JetBrains. *"WebStorm: The smartest JavaScript IDE", jetbrains.com.* URL: https://www.jetbrains.com/webstorm (visitado 25-06-2021).

[10] JetBrains. *"IntelliJ IDEA: The Capable and Ergonomic Java IDE by JetBrains", jetbrains.com.* URL: https://www.jetbrains.com/idea (visitado 25-06-2021).

[11] Nordic. *"Explore our product portfolio", nordicsemi.com.* URL: https://www.nordicsemi.com/Software-and-tools (visitado 25-06-2021).

[12] Mongo. *"Managed MongoDB Hosting | Database-as-a-service | MongoDB", mongodb.com.* URL: https://www.mongodb.com/es/cloud/atlas (visitado 25-06-2021).

[13] Mongo. *"Compass | MongoDB", mongodb.com.* URL: https://www.mongodb.com/es/products/compass (visitado 25-06-2021).

[14] Centro de Transferencia de Tecnología - Forja CTT. *"Compass | MongoDB", github.com.* URL: https://github.com/ctt-gob-es/clienteafirma/ (visitado 25-06-2021).

[15] Signaturit Media. *"eIDAS: Reglamento de la Unión Europea de firma electrónica", blog.signaturit.com.* 2020. URL: https://blog.signaturit.com/es/eidas-nuevos-tiempos-para-la-firma-electronica-en-europa (visitado 03-06-2021).

[16] Eur-lex. *"REGLAMENTO (UE) No 910/2014 DEL PARLAMENTO EUROPEO Y DEL CONSEJO", eur-lex.europa.eu.* 2014. URL: https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX:32014R0910 (visitado 03-06-2021).

[17] Autoridad de Certificación Asociación Nacional de Fabricantes de España. *"DIGITALIZACIÓN Y AUTOMATIZACIÓN DE EMPRESAS", anf.es.* URL: https://www.anf.es/digitalizacion-y-automatizacion-de-empresas/#egoverment (visitado 03-06-2021).

[18] Comisión Europea. *"Commission proposes a trusted and secure Digital Identity", ec.europa.eu.* 2021. URL: https://ec.europa.eu/commission/presscorner/detail/en/ip_21_2663 (visitado 22-06-2021).

[19] Portal de la Administración Electrónica. *"Sistema europeo de reconocimiento de identidades electrónicas - eIDAS", administracionelectronica.gob.es.* URL: https://administracionelectronica.gob.es/ctt/eidas (visitado 09-06-2021).

[20] Agencia Estatal Boletín Oficial del Estado. *"BOE.es - BOE-A-2021-5032 Real Decreto 203/2021, de 30 de marzo, por el que se aprueba el Reglamento de actuación y funcionamiento del sector público por medios electrónicos.", boe.es.* 2021. URL: https://www.boe.es/buscar/act.php?id=BOE-A-2021-5032 (visitado 11-06-2021).

[21] P. Wright. *"La delegación de firma", viafirma.com.* 2021. URL: https://www.viafirma.com/blog-xnoccio/es/delegacion-de-firma (visitado 10-06-2021).

[22] Agencia Estatal Boletín Oficial del Estado. *"Disposición 10566 del BOE núm. 236 de 2015", boe.es.* 2015. URL: https://www.boe.es/boe/dias/2015/10/02/pdfs/BOE-A-2015-10566.pdf (visitado 11-06-2021).

[23] A. Palomar y J. Fuertes. *"Delegación de firma - Competencias", vlex.es.* URL: https://vlex.es/vid/delegacion-firma-427628358 (visitado 10-06-2021).

[24] MDN Web Docs. *"Native messaging - Mozilla | MDN", developer.mozilla.org.* URL: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messaging (visitado 16-06-2021).

[25] Signer Digital. *"Digital Signing and Digital Signature based Authentication from Web Browser", help.signer.digital.* URL: https://help.signer.digital/digital_signing_and_authentication_from_web_browser.htm (visitado 11-06-2021).

[26] Riigi Infosüsteemi Ameti. *"Web eID: electronic ID smart cards on the Web", web-eid.eu.* URL: https://web-eid.eu/ (visitado 24-06-2021).

[27] Microcosm. *"IT Security Hardware", microcosm.com.* URL: https://www.microcosm.com/it-security-hardware (visitado 12-06-2021).

[28] Grupo 2000. *"FirmaCloud. La solución más segura para utilizar y gestionar tu Certificado Digital", grupo2000.es.* URL: https://www.grupo2000.es/firmacloud-la-solucion-mas-segura-para-utilizar-tu-certificado-digital/ (visitado 12-06-2021).

[29] Grupo 2000. *"La aplicación de Firma Electrónica para tu empresa", firmafy.com.* URL: https://firmafy.com/ (visitado 12-06-2021).

[30] Viafirma. *"Viafirma Inbox · Portafirmas para firma digital", viafirma.com.* URL: https://www.viafirma.com/inbox/es/ (visitado 12-06-2021).

[31] Viafirma. *"Viafirma Fortress - Gestiona y controla tus firmas digitales", viafirma.com.* URL: https://www.viafirma.com/fortress/es/ (visitado 12-06-2021).

[32] Docuten. *"What is a delegated signature?", docuten.com.* URL: https://docuten.com/en/faqs/what-is-the-delegated-signature/ (visitado 12-06-2021).

[33] Secretaría General de Administración Digital (SGAD). *"Catálogo de servicios de administración digital", administracionelectronica.gob.es.* URL: https://administracionelectronica.gob.es/pae_Home/dam/jcr:736d93af-5aef-457c-bad8-f5496ffa1734/Catalogo-servicios-administracion-digital-version-2018.pdf (visitado 14-06-2021).

[34] Ministerio de Justicia. *"Autenticación - Sede Electrónica", sede.mjusticia.gob.es.* URL: https://sede.mjusticia.gob.es/es/informacion-ayuda/certificado-digital/autenticacion (visitado 14-06-2021).

[35] clave.gob.es. *"¿Qué es la firma centralizada?", clave.gob.es.* URL: https://clave.gob.es/clave_Home/dnin/queEs.html (visitado 14-06-2021).

[36] clave.gob.es. *"Realización de la Firma - Cl@ve Firma - Cl@ve", clave.gob.es.* URL: https://clave.gob.es/clave_Home/dnin/realizando-la-firma.html (visitado 14-06-2021).

[37] Ministerio de Hacienda y Administraciones Públicas Dirección de Tecnologías de la Información y las Comunicaciones. *"Presentación de la Suite @firma", administracionelectronica.gob.es.* URL: http://www.administracionelectronica.gob.es/dam/jcr:00df46e5-7b1f-48cd-9cb2-d71b197ca383/Suite@Firma.pdf (visitado 14-06-2021).

[38] Portal de la Administración Electrónica. *"PAe - CTT - General - Cliente de firma electrónica de @firma", administracionelectronica.gob.es.* URL: https://administracionelectronica.gob.es/ctt/clienteafirma#.YL40yzqxUWQ (visitado 14-06-2021).

[39] *"ctt-gob-es/fire: Sistema de firma integral FIRe", github.com.* URL: https://github.com/ctt-gob-es/fire (visitado 14-06-2021).

[40] Portal de la Administración Electrónica. *"PAe - CTT - área de Descargas - FIRe - Solución Integral de Firma Electrónica", administracionelectronica.gob.es.* URL: https://administracionelectronica.gob.es/ctt/fire/descargas (visitado 14-06-2021).

[41] Portal de la Administración Electrónica. *"PAe - CTT - General - AutenticA: El repositorio horizontal de usuarios de las Administraciones Publicas", administracionelectronica.gob.es.* URL: https://administracionelectronica.gob.es/ctt/autentica#.YL45YTqxUWQ (visitado 14-06-2021).

[42] Cuerpo Nacional de Policía. *"Firma Electrónica", dnielectronico.es.* URL: https://www.dnielectronico.es/PortalDNIe/PRF1_Cons02.action?pag=REF_1069&id_menu=27 (visitado 14-06-2021).

[43] Comisión Técnica de apoyo a la implantación del DNE electrónico. *"Guia de referencia basica app de firma", dnielectronico.es.* 2006. URL: https://www.dnielectronico.es/PDFs/anexo_guia.pdf (visitado 14-06-2021).

[44] F. Morgner. *"Welcome to the Virtual Smart Card Architecture documentation!", frankmorgner.github.io.* URL: https://frankmorgner.github.io/vsmartcard/index.html (visitado 03-06-2021).

[45] F. Morgner. *"Virtual Smart Card - vsmartcard", frankmorgner.github.io.* URL: https://frankmorgner.github.io/vsmartcard/virtualsmartcard/README.html (visitado 17-06-2021).

[46] The Web eID project. *"web-eid/web-eid-system-architecture-doc: The Web eID project enables usage of European Union electronic identity smart cards for secure authentication and digital signing of documents on the web using public-key cryptography", github.com.* URL: https://github.com/web-eid/web-eid-system-architecture-doc (visitado 22-06-2021).

[47] W3C. *"XML Advanced Electronic Signatures (XAdES)", w3.org.* 2003. URL: https://www.w3.org/TR/XAdES/ (visitado 20-06-2021).

Apéndice A
Manual de la Aplicación Nativa. Instalación y uso

A.1. General

Los **requisitos previos** son los siguientes:

- **Python y pip**: la aplicación se ha desarrollado con Python 3.9.2 y probado con Python 3.9.2 y 3.8, sin embargo, debe funcionar con cualquier versión de Python superior a 3.7 y el gestor de paquetes *pip*. Las actualizaciones de Python se deben hacer antes de la instalación de las librerías, pues cualquier cambio implicará repetir el proceso.

- **Navegador**: la configuración de la aplicación nativa se ha probado que funciona correctamente en **Firefox** únicamente.

- **Certificados**: los certificados probados han sido con extensiones *.p12* y *.pfx* con los algoritmos de firma *sha256withrsa*. En principio, debería de funcionar con otros algoritmos, pero es de destacar que:

  - Los certificados con **SHA1** no funcionan, pues la librería *PyJwt* no los soporta para la firma de tokens.
  - Los certificados con **ECDSA** no funcionan ya que *Cryptography* no deja firmar con ellos.

  No obstante, hay certificados de prueba, cuyo uso se explica en la sección de uso.

Las **librerías** de Python se deben instalar previamente:

- **Cryptography**[37]: aunque esta librería venga incluida en la versión de Python de algunos sistemas Linux, probablemente sea necesario actualizar, pues se necesita la **versión 3.1 o superior**. Para comprobar la versión de la librería, se puede usar *pip freeze | grep cryptography*, y para actualizar, usar *python3 -m pip install -U cryptography* o *python3 -m pip3 install -U cryptography*
- **PyCryptodome**[38]
- **Aenum**[39]
- **Dotenv**[40]
- **PyJWT**[41]
- **Requests**[42]
- **Validators**[43]

A.2. Windows

La aplicación nativa se ha probado en Windows 10 64 bits. Tras instalar las librerías y comprobar que la versión de Python es correcta:

1. Ejecutar el script *web_eid_bat_setup.bat*, que creará el fichero */app/webeidPython.bat*, usado para ejecutar el script principal de la aplicación (*/app/webeidPython.py*).
2. Ejecutar el script *web_eid_json_setup.ps1* con powershell, que modificará el fichero */app/webeidPython.json*, cambiando el valor de *path* por la ruta completa del fichero creado en el paso previo.
3. Ejecutar el script *web_eid_modified_regedit_setup.bat*. Pedirá permisos de administrador para modificar el editor de registros de Windows, añadiendo las claves necesarias para permitir la comunicación entre la extensión y la aplicación nativa.

[37] https://cryptography.io/en/latest/
[38] https://pycryptodome.readthedocs.io/en/latest/
[39] https://pypi.org/project/aenum/
[40] https://pypi.org/project/python-dotenv/
[41] https://pyjwt.readthedocs.io/en/stable/
[42] https://pypi.org/project/requests/
[43] https://pypi.org/project/validators/

4. Cambiar la primera línea del fichero *app/webeidPython.py* a *#!/usr/bin/env python*

Si tras estos pasos existen errores, se debe comprobar:

1. Que el fichero *webeidPython.bat* se ha creado en la carpeta */app*.
2. Que dentro de *webeidPython.bat*, el argumento de *call python* es la ruta completa a *webeidPython.py* de la forma *X:\ruta\al\fichero*
3. Que el valor del campo *path* de *webeidPython.json* es la ruta completa a *webeidPython.bat* de la forma *"X:\\ruta\\al\\fichero"*
4. Los valores de las claves en el editor de registros de Windows:
   a) Escribir *regedit* en la barra de búsqueda general de Windows.
   b) Ir a las rutas *HKEY_LOCAL_MACHINE\SOFTWARE\Mozilla\NativeMessagingHosts* y *HKEY_CURRENT_USER\SOFTWARE\Mozilla\NativeMessagingHosts*.
   c) Comprobar que en ambas hay una carpeta (clave) llamada *webeidPython*.
   d) Comprobar que el valor de dicha clave es la ruta completa a *webeidPython.json* de la forma *X:\ruta\al\fichero*.

Si tras dichas correcciones y comprobaciones continúa sin funcionar, algunas fuentes de ayuda son:

- El fichero *firefox_extension_setup.md*, que contiene las instrucciones del ejemplo de paso de mensajes nativo de la web de MDN.
- El propio artículo de MDN Developers sobre paso de mensajes nativo [44], que contiene tanto instrucciones de instalación como resolución de problemas y preguntas frecuentes.
- El proyecto original de la aplicación nativa de Web-eID[45], con instrucciones de instalación de la app original.
- El proyecto original de la extensión de Web-eID[46], con instrucciones de instalación de la extensión original.

[44] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messaging
[45] https://github.com/web-eid/web-eid-app
[46] https://github.com/web-eid/web-eid-webextension

- Finalmente, en la carpeta */app/logs* se guardan tanto mensajes de información, como de debug y de errores. Puede ser de utilidad para determinados problemas.

A.3. Linux

La aplicación nativa se ha probado en Ubuntu 20.04 LTS y 18.04 LTS, aunque en este último quizás sea necesario actualizar la versión de Python y de node.

1. Dar permisos de ejecución al script principal de la aplicación nativa con *chmod +x app/webeidPython.py*
2. Instalar *jq* con el comando *sudo apt-get install jq*.
3. Comprobar que existe la carpeta */usr/lib/mozilla/native-messaging-hosts*, y en caso de que no exista, ejecutar el comando *sudo mkdir /usr/lib/mozilla/native-messaging-hosts*.
4. Ejecutar el script *web_eid_setup_linux.sh* con permisos de administrador. Esto localizará el script principal de la aplicación nativa, copiará la ruta en el campo *path* de *webeidPython.json* y copiará dicho archivo a la carpeta */usr/lib/mozilla/native-messaging-hosts*. Para ello, ejecutar *sudo ./web_eid_setup_linux.sh*
5. Modificar la primera línea de *webeidPython.py* a *#!/usr/bin/env python3* o a la localización del ejecutable de Python3.7 (o versión superior) en la máquina.

Si tras estos pasos existen errores, se debe comprobar:

- Que los permisos de *webeidPython.py* incluyen los permisos de ejecución por parte del usuario actual.
- Que el archivo *webeidPython.json* está en la carpeta */usr/lib/mozilla/native-messaging-hosts*.
- Que el valor del campo *path* en *webeidPython.json* es la ruta completa a *webeidPython.py* entre comillas dobles.
- La ruta del ejecutable de Python3: si al escribir en la terminal */usr/bin/env python3* o la ruta indicada en la primera línea de *webeidPython.py* no provoca que se inicie la consola

de python, significa que la ruta no es correcta y por tanto el script principal no puede ser ejecutado con dicho comando. En ese caso se debe comprobar dónde está instalado *python3* y modificar correctamente la primera línea de *webeidPython.py* con la ruta precedida de *#!*

Si tras dichas correcciones y comprobaciones continúa sin funcionar, algunas fuentes de ayuda son:

- El fichero *firefox_extension_setup.md*, que contiene las instrucciones del ejemplo de paso de mensajes nativo de la web de MDN.
- El propio artículo de MDN Developers sobre paso de mensajes nativo [47], que contiene tanto instrucciones de instalación como resolución de problemas y preguntas frecuentes.
- El proyecto original de la aplicación nativa de Web-eID[48], con instrucciones de instalación de la app original.
- El proyecto original de la extensión de Web-eID[49], con instrucciones de instalación de la extensión original.
- Finalmente, en la carpeta */app/logs* se guardan tanto mensajes de información, como de debug y de errores. Puede ser de utilidad para determinados problemas.

A.4. Uso

La aplicación nativa puede utilizarse tanto con los certificados de prueba como con los certificados personales (con extension *.pfx* o *.p12*).

1. Renombrar el fichero *.env.dist* a *.env*
2. Para uso de certificados de prueba:
    a) Poner el valor de la variable *USE_PERSONAL_CERT* a *False*.

[47] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messaging
[48] https://github.com/web-eid/web-eid-app
[49] https://github.com/web-eid/web-eid-webextension

b) Poner el valor de la variable *TEST_CERT_NAME* con uno de los nombres de los certificados de prueba de la carpeta *certs* (únicamente nombre del fichero y extensión). Mencionar que el único que funciona correctamente es *PruebaEmpleado4Activo.p12*, pero el resto se puede utilizar para comprobar que en los logs saltan errores con las librerías mencionadas.

3. Para uso de certificados personales:

   a) Poner el valor de la variable *USE_PERSONAL_CERT* a *True*.

   b) Poner el valor de la variable *PERSONAL_CERT_PATH* con la ruta completa al certificado que se quiera utilizar, entre comillas dobles.

   c) Poner el valor de la variable *PERSONAL_CERT_PW* con la contraseña del certificado entre comillas dobles.

4. Los registros de uso de certificados, así como los archivos que contienen los metadatos de los documentos firmados, se almacenan en la carpeta */app/registry*.

Nota: las líneas del fichero *.env* no deben contener espacios entre nombres, valores de variables y signos. Un ejemplo de uso correcto es *PERSONAL_CERT_PW="password"*.

Apéndice B
Manual de la Extensión. Instalación.

Los **requisitos previos** son los siguientes:

- **Node** (nota: se ha utilizado la versión 12.19.0)
- **Npm** (nota: se ha utilizado la versión 6.14.8)
- **Navegador**: todos los componentes se han probado únicamente en **Firefox**.

Nota: en algunos sistemas linux, es necesario actualizar la versión de Node y Npm[50].

Los **pasos de instalación** son:

1. Ir a la carpeta raíz del proyecto
2. Ejecutar el comando *npm install* para instalar las dependencias.
3. Comprobar que el valor de NATIVE_APP_NAME en *src/config.ts* es igual que el valor del campo *name* en el manifest de la aplicación nativa (*/app/webeidPython.json*).
4. Ejecutar el comando *npm run clean build package* para construir la extensión.
5. Abrir el navegador Firefox e ir a la ruta *about:debugging#/runtime/this-firefox*
6. Seleccionar *cargar complemento temporal.*
7. Seleccionar el archivo *manifest.json* de la carpeta */dist/firefox* dentro del proyecto de la extensión.

[50] https://askubuntu.com/questions/426750/how-can-i-update-my-nodejs-to-the-latest-version

Apéndice C
Manual de la Aplicación web modificada. Instalación y uso.

C.1. Uso remoto

Esta aplicación está disponible de forma remota en el cloud de heroku, por lo que es utilizable sin instalación alguna accediendo a https://webeidspringexamplemodified.herokuapp.com

C.2. Uso local

Los **requisitos generales** son simplemente disponer de un jdk y jre de Java, tener la variable *JAVA_HOME* apuntando a la carpeta del jdk instalado y descargar ngrok[51]. Al igual que todos los componentes, solamente se ha probado en **Firefox**.

1. Ejecutar el comando *ngrok http 8080*. Esto redireccionará a *localhost* las peticiones a las urls generadas.

2. Cambiar el valor de *local-origin* en el archivo *src/main/resources/application.yaml* a la dirección https generada por ngrok.

3. Dentro de la carpeta raíz del proyecto, ejecutar el comando *./mvnw spring-boot:run*

    - En linux puede requerir previamente otorgar permisos de ejecución a *mvnw* con *chmod +x ./mvnw*

[51] https://ngrok.com/download

Apéndice D
Manual del Servidor de Registros. Instalación y uso.

D.1. Uso remoto

Al igual que la aplicación web modificada, el servidor de registros está disponible en la nube, en el cloud de heroku, por lo que es utilizable sin instalación alguna accediendo al siguiente enlace https://eidrecordserver.herokuapp.com

D.2. Uso local

Los **requisitos generales** son simplemente disponer de un jdk y jre de Java, tener la variable *JAVA_HOME* apuntando a la carpeta del jdk instalado y descargar ngrok[52]. Al igual que todos los componentes, solamente se ha probado en **Firefox**.

Nota: el envío de correos electrónicos solo se puede probar en local si no se dispone de la contraseña de la cuenta de gmail del servidor de registros, por tanto es necesario descativar el envío de correos en el archivo */src/main/java/org/webeid/eidRecordServer/web/rest/EidRecordController.java*

1. Ejecutar el comando *ngrok http 8080*. Esto redireccionará a *localhost* las peticiones a las urls generadas.

2. Cambiar el valor de *local-origin* en el archivo *src/main/resources/application.yaml* a la dirección https generada por ngrok.

3. En el proyecto de la extensión, cambiar el valor de la constante *recordUrl* tanto en *src/background/actions/authenticate.ts* como en *src/background/actions/sign.ts* por la nueva

[52] https://ngrok.com/download

dirección https generada por ngrok. Esto se puede hacer comentando la línea con la dirección de heroku, quitando el comentario de la línea de debajo y cambiando el valor de la url de ngrok.

4. En el proyecto de la aplicación nativa, cambiar el valor de la variable *post_url* en el archivo *app/src/controller/command_handler/postRecord.py* por la url de ngrok.

5. Volviendo al proyecto del servidor de registros, en la carpeta raíz, ejecutar el comando *./mvnw spring-boot:run*

    - En linux puede requerir previamente otorgar permisos de ejecución a *mvnw* con *chmod +x ./mvnw*

Apéndice E
Manual de usuario de la solución integral

Este apéndice va enfocado a **mostrar el flujo de utilización de la prueba de concepto**, como lo haría un usuario en el día a día. Aunque se haya explicado brevemente el uso de cada componente en los apéndices previos, aquí se materializará mostrando los pasos a seguir y las pantallas y resultados obtenidos. Nótese que en este apéndice **se hará uso de las versiones remotas de las dos aplicaciones web del proyecto**, pues no hay diferencias de uso con las versiones locales y los pasos para levantar en local ambas aplicaciones se explican los apéndices previos. Finalmente, **se da por hecho que se han realizado los pasos de instalación y configuración tanto de la extensión como de la aplicación nativa.**

Para el primer ejemplo de uso integral, se utilizará el **certificado de prueba.** Para ello, el archivo de entorno de la aplicación nativa (*.env*) debe estar como sigue.

```
TEST_CERT_NAME="PruebaEmpleado4Activo.p12"
TEST_CERT_PW="Giss2016"

PERSONAL_CERT_PATH="C:\\Path\\to\\personal\\cert.p12"
PERSONAL_CERT_PW="personal_cert_pw"

USE_PERSONAL_CERT=False
```

Figura 23: Archivo *.env* para uso de certificado de prueba.

Una vez configurado, se debe acceder al navegador, cargar la extensión como se indica en su apéndice de instalación y acceder tanto a la página del servidor de registros[53] como a la del ejemplo modificado de la aplicación de autenticación y firma[54]. De primeras puede tardar un

[53] https://eidrecordserver.herokuapp.com/
[54] https://webeidspringexamplemodified.herokuapp.com/

tiempo adicional en cargar, pues heroku *duerme* las aplicaciones si no se usan en media hora. En la esquina superior derecha debe aparecer un icono similar al de la UMA modificado (el icono de la extensión).

Figura 24: Icono de la extensión en la barra de extensiones de Firefox.

No es relevante la página el la que se inicie sesión en primer lugar, pero en este caso se hará primero en el servidor de registros, y posteriormente en la aplicación de ejemplo. En la página del servidor de registro, se **debe bajar hasta encontrar el botón *authenticate* y pulsarlo para iniciar sesión**.

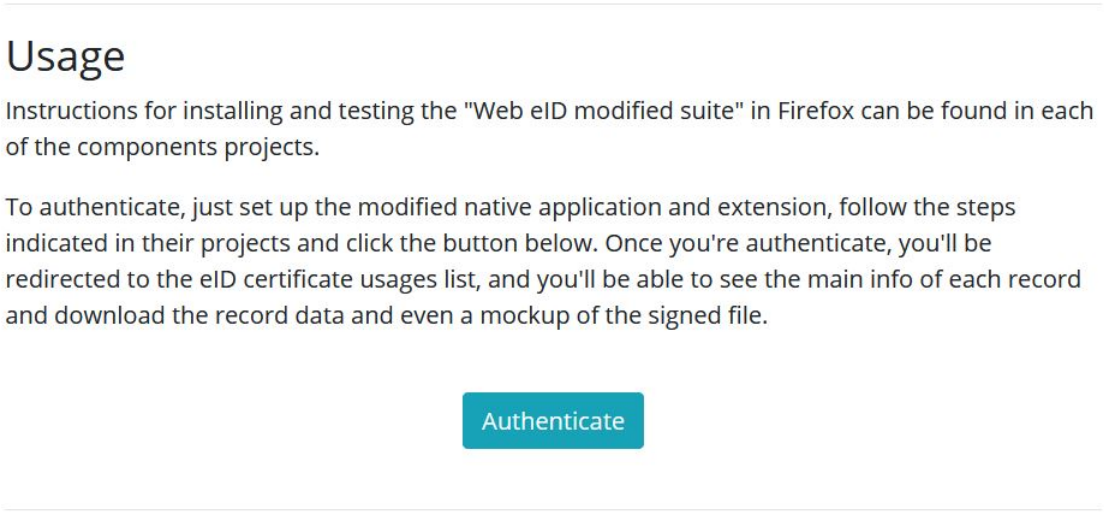

Figura 25: Botón de inicio de sesión en el servidor de registros

Una vez se inicia sesión, debe aparecer una lista con los registros de uso. Puesto que este certificado de prueba se empaqueta con el proyecto, es posible que otras personas lo hayan utilizado, pero debido a que la propia autenticación en el servidor se registra, debe aparecer en primer lugar el registro de autenticación en el servidor. Además, el estado de las notificaciones dependerá de las últimas pruebas que se hayan hecho con él.

Figura 26: Página de lista de registros del servidor de registros.

Nota: al ser un certificado de prueba con una dirección de correo electrónico no válida, no se puede comprobar que llegan los correos correctamente. Esta funcionalidad se comprobará en ejemplo de uso con certificado personal.

A continuación, en la página del ejemplo modificado, bajar hasta el botón de autenticación, similar al del servidor de registros.

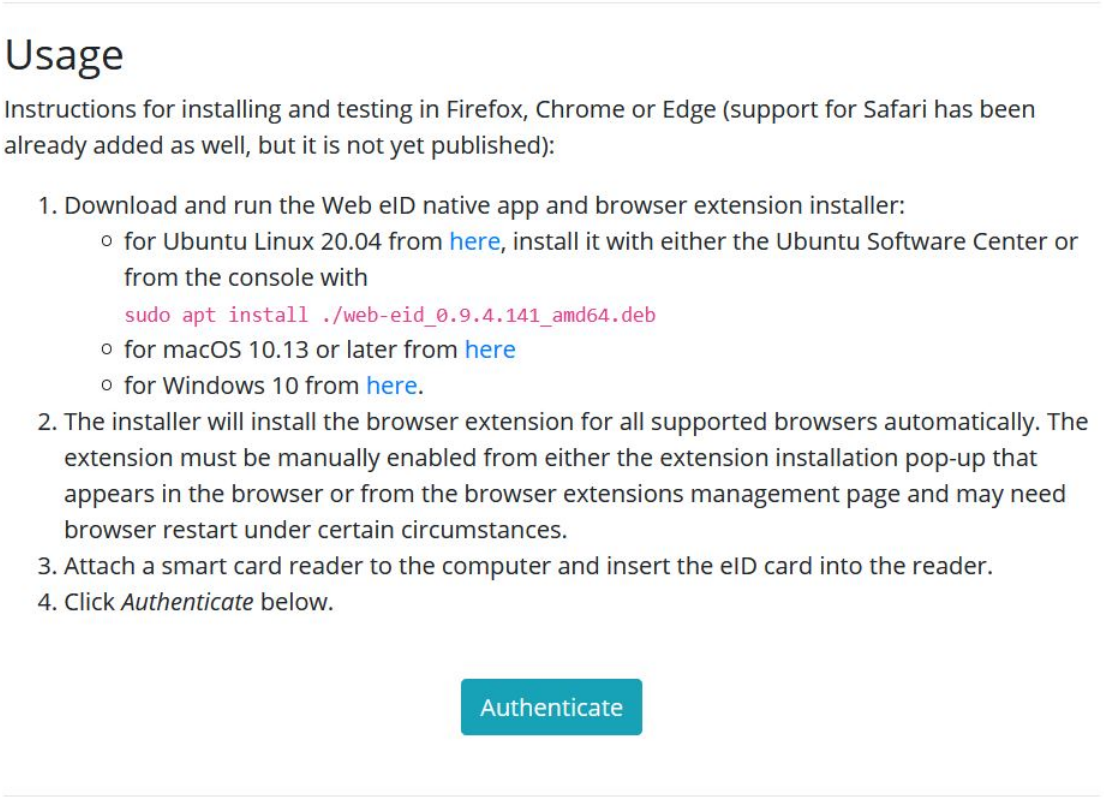

Figura 27: Botón de inicio de sesión en el ejemplo modificado de Web-eID.

Al iniciar sesión en esta página, se muestra un fichero de texto para firmar, así como un botón para cerrar sesión, que nos llevará de vuelta a la página inicial, y al pulsar el botón *Sign document*, tras un breve instante, aparecerá un botón que permite descargar el contenedor de firma, y se indica que todo el proceso se ha completado con éxito.

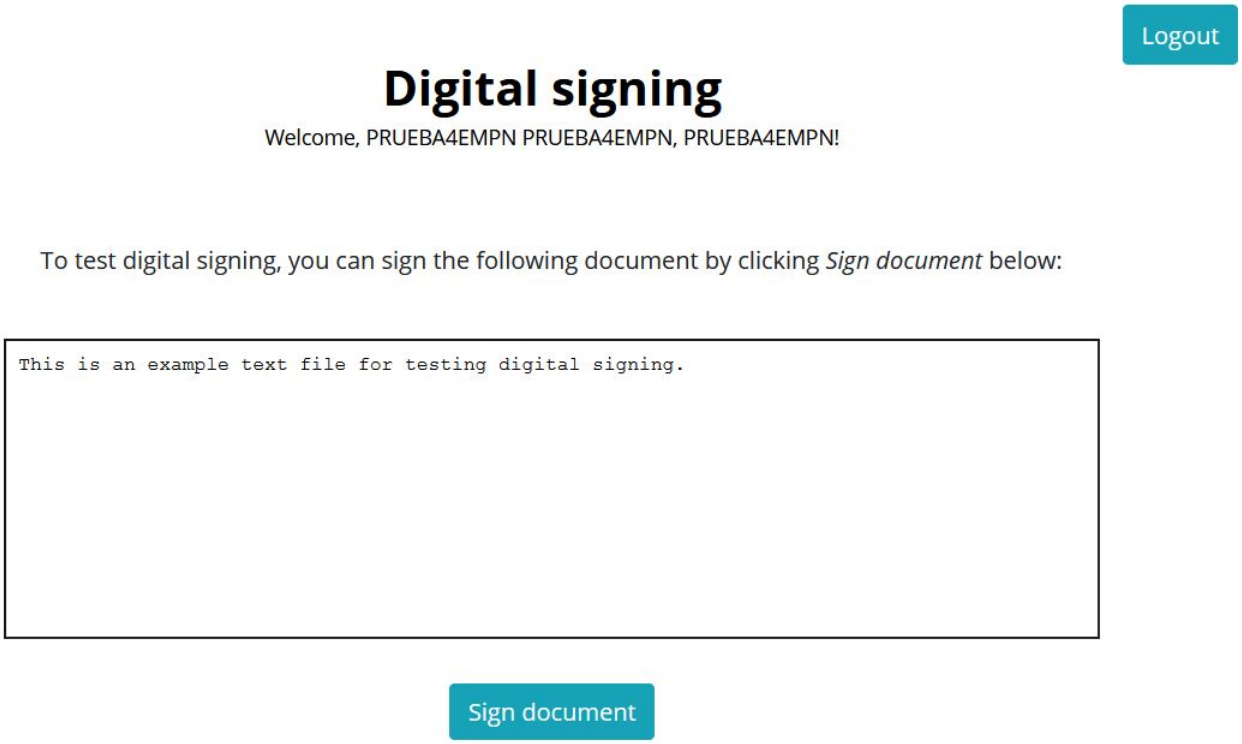

Figura 28: Página de bienvenida del ejemplo modificado de Web-eID.

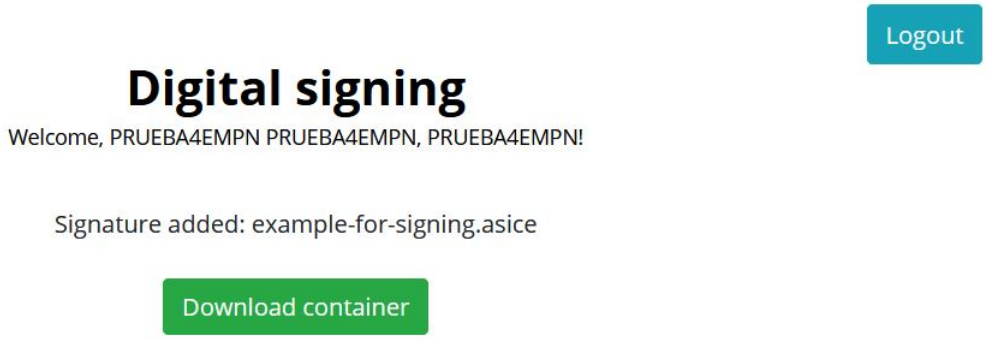

Figura 29: Resultado exitoso de firma en el ejemplo modificado de Web-eID.

Para comprobar que la firma se ha hecho con éxito, se puede descargar el contenedor y subirlo a la herramienta de comprobación online de *nowina*[55]. Subiendo el archivo descargado en el apartado *Signed File* y pulsando el botón *submit*, se puede apreciar que la única advertencia es que el contenedor no ha sido firmado con un dispositivo hardware (QSCD) pero por lo demás pasa la validación completa.

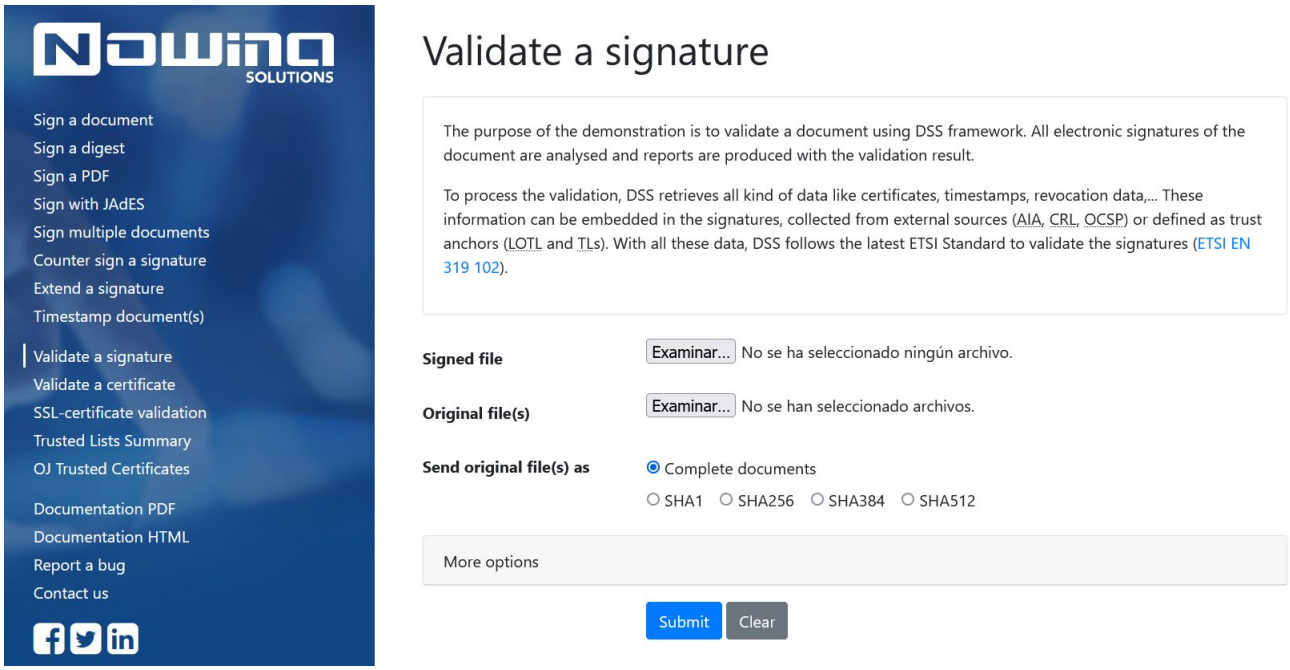

Figura 30: Página principal de la herramienta de validación de Nowina.

[55] https://dss.nowina.lu/validation

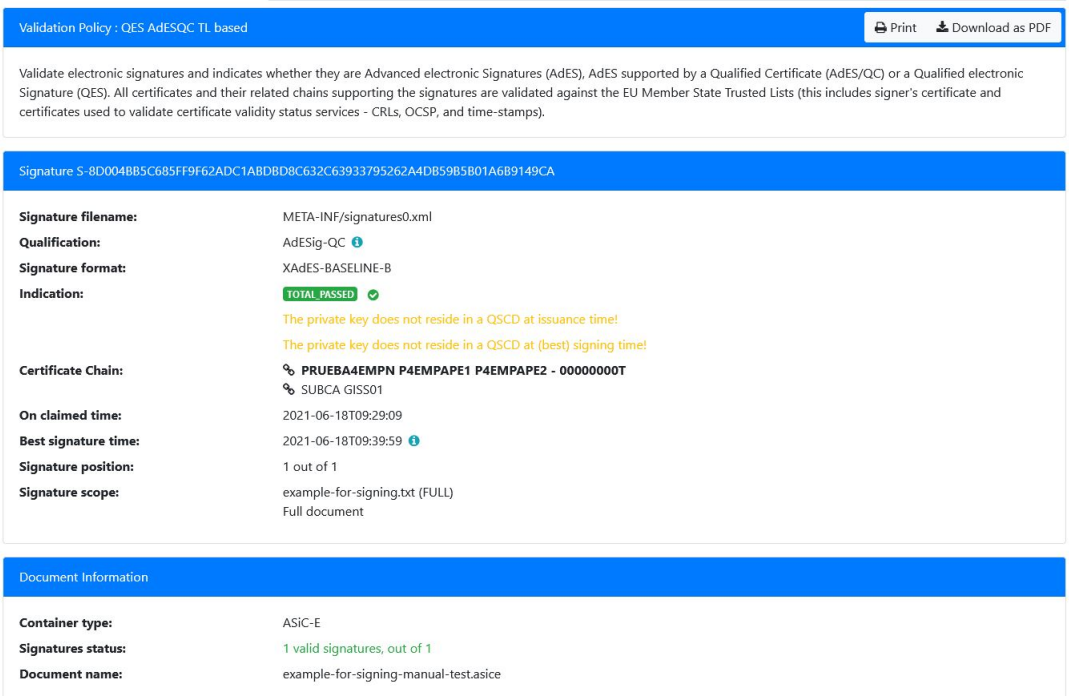

Figura 31: Análisis de la firma del contenedor descargado.

Volviendo a la lista de registros del servidor de registros, si se recarga la página se apreciará que han aparecido dos nuevos registros, uno por la autenticación en la aplicación de ejemplo y otro por la firma. En este último, se ofrece la posibilidad de descargar el mock del fichero obtenido.

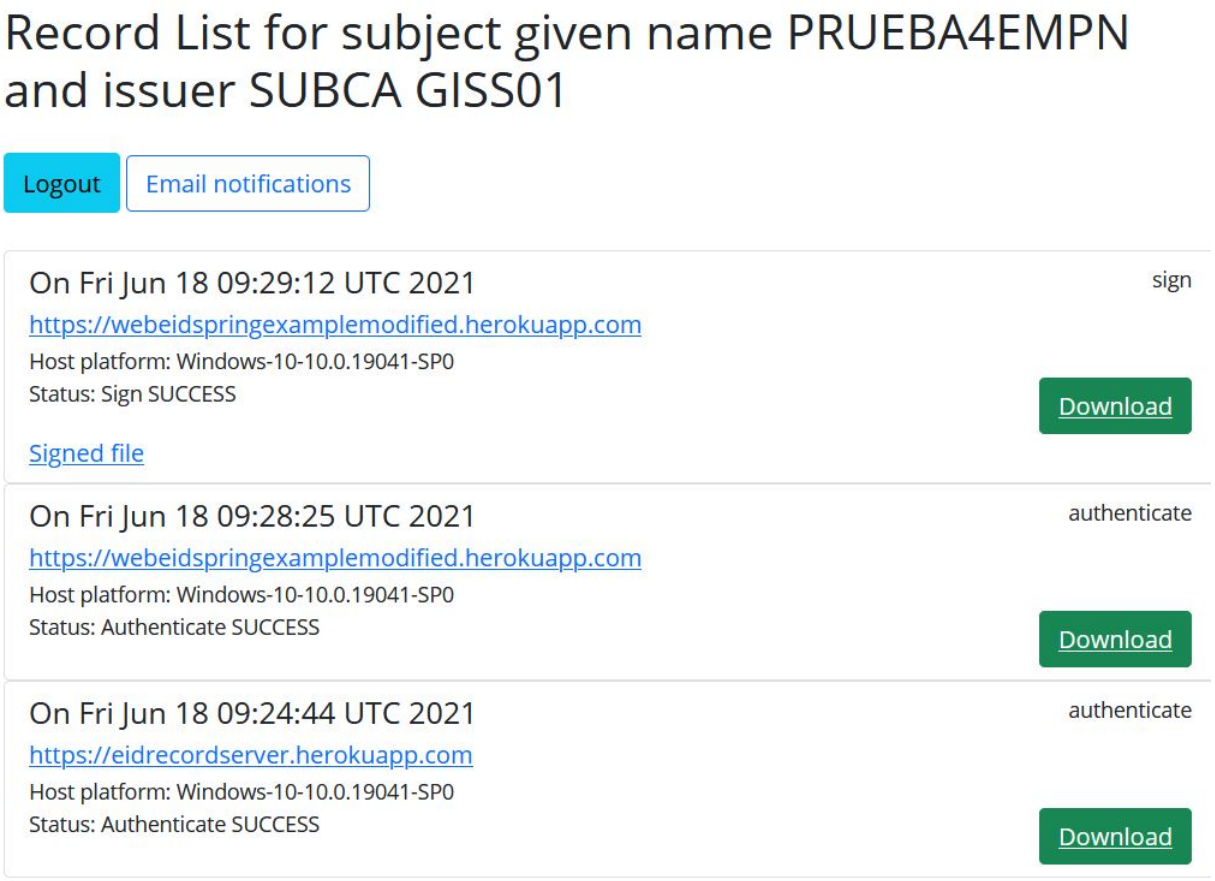

Figura 32: Lista de registros tras la autenticación y la firma en el ejemplo modificado de Web-eID.

Se puede probar a descargar, por ejemplo, el registro situado abajo del todo (el primer registro de autenticación) y el *mock* del fichero firmado.

Figura 33: Contenido del registro de autenticación descargado desde el servidor de registros (abierto en Firefox).

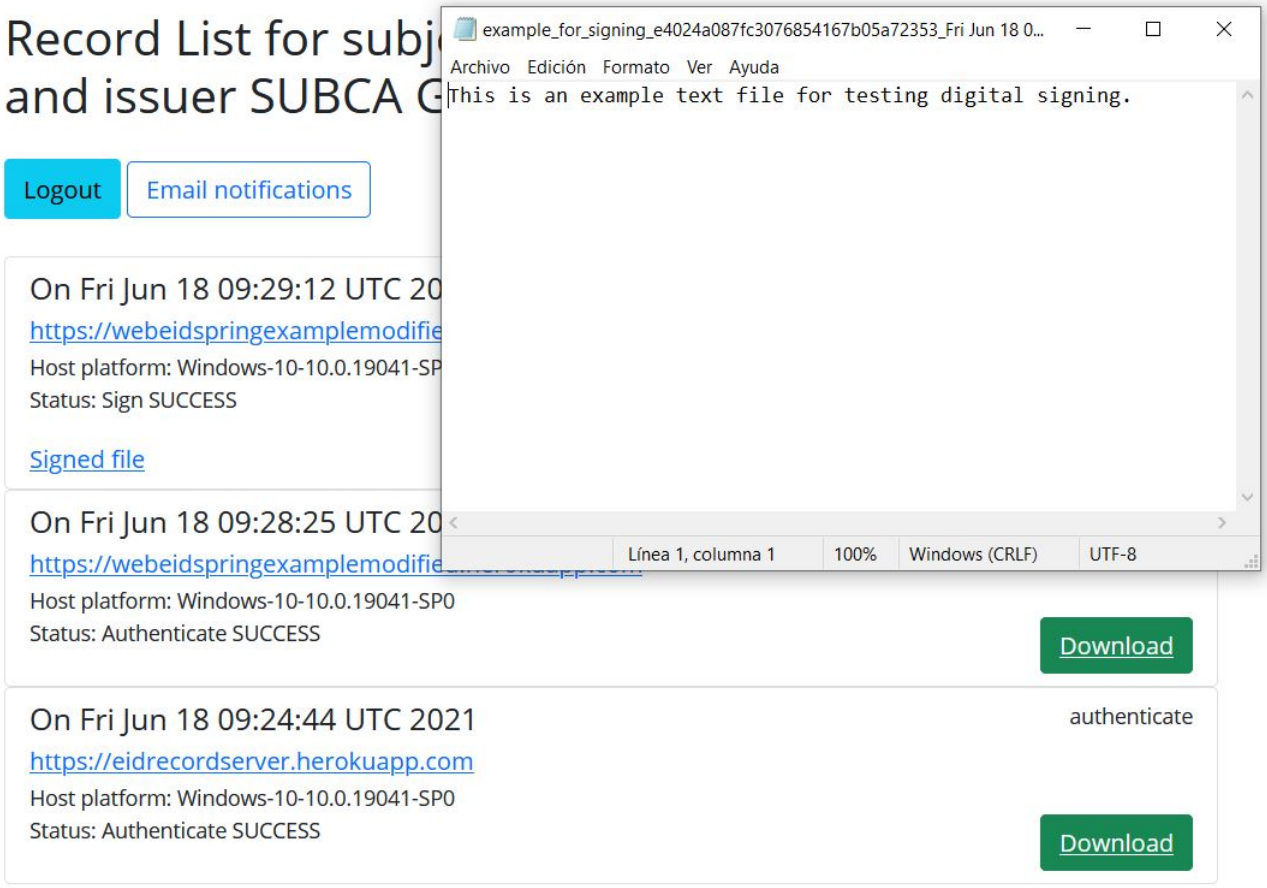

Figura 34: *Mock* del fichero de firma descargado desde el servidor de registros.

Todo este proceso, por supuesto, también se ha registrado en local. Accediendo a la carpeta *app/registry* dentro del proyecto de la aplicación nativa, se aprecia que han aparecido cuatro archivos: tres de ellos correspondientes a cada registro y uno correspondiente al contenedor de firma. Se pueden abrir, por ejemplo, con Notepad++.

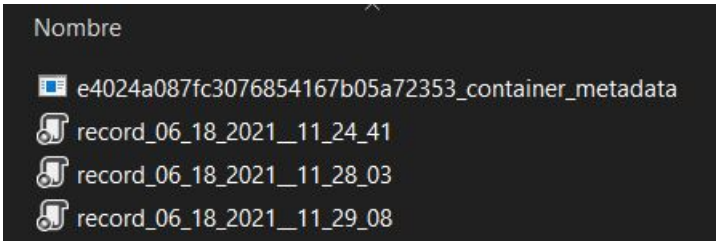

Figura 35: Registros de uso del certificado en local

Figura 36: Contenido del registro de firma almacenado en local

Para el uso del **certificado personal**, se debe modificar el fichero de entorno de acuerdo a lo indicado en el manual de la aplicación nativa. Al ser un certificado personal, no estará almacenado en la base de datos, así que tendrá activadas las notificaciones por correo. Iniciando sesión en el servidor de registros se puede comprobar que sólo aparecen los registros del certificado personal, y que se ha recibido un correo electrónico (si el certificado tiene el correo electrónico).

Figura 37: Lista de registros de uso del certificado personal.

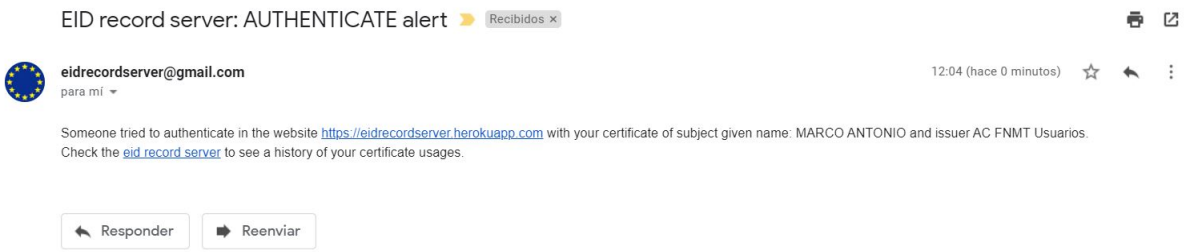

Figura 38: Correo electrónico de aviso de autenticación.

Si se mantienen las notificaciones activadas y se accede y firma en el ejemplo modificado de Web-eID, se recibirán dos nuevos correos electrónicos. Además, en la carpeta local deben haber aparecido nuevos registros de uso.

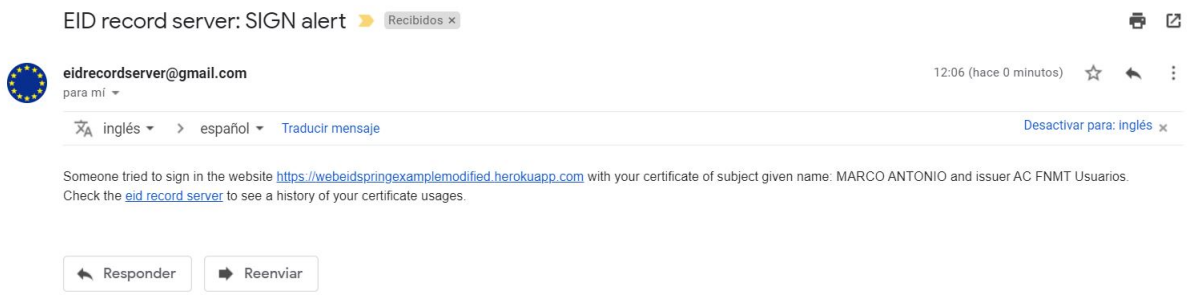

Figura 39: Correo electrónico de aviso de firma.

Apéndice F
Problemas en el diseño y desarrollo de la prueba de concepto

En este apéndice se exponen los principales problemas encontrados en el proceso de desarrollo de la prueba de concepto.

En primer lugar, la fase de investigación de la solución de diseño. Comenzando con **Vsmartcard**, **el primer problema es la claridad del repositorio**, pues da las opciones de clonar el repositorio y actualizar todo con *git clone https://github.com/frankmorgner/vsmartcard.git; cd vsmartcard; git submodule update –init –recursive* o de descargar el zip de la *release*. Si se escoge el segundo método, el tercer proyecto o *solución* de Visual Studio (*devMSI*) sale vacío.

Además, supone que el usuario conoce cómo compilar un controlador con la herramienta *devcon*, y solo vienen indicaciones sobre cómo compilarlo desde *Visual Studio 2015*. También da a entender que el instalador (con extensión *.msi*) se descarga con los binarios de la **release** y no es así. Finalmente, tampoco menciona que el archivo *wudf_update* debes copiarlo desde tu carpeta del sistema, pues depende de la versión particular de Windows 10.

Respecto a las **herramientas que pide**, utiliza Visual Studio 2015, wdk 10 y una versión antigua de WiX Toolset (3.10), lo cual ha causado varios problemas:

- Problemas de sintaxis en Visual Studio 2019.
- En Visual Studio 2019 además pide actualizar a la versión de WiX 3.11.
- Problemas de tipo MIDL en el proyecto del driver (*BixVReader*) al encontrar el archivo *wudfddi.idl*.

Sin embargo, aunque ocurran errores, el instalador finalmente se compila y al ejecutarlo, aparece un dispositivo desconocido en el administrador de dispositivos de Windows. Al intentar actualizar el controlador con el *dll* proporcionado en los binarios, resulta en un error de *devcon.exe*. Además, al ejecutar la herramienta de OpenSC[56] (*opensc-tool -l*) que lista los lectores de tarjeta, no aparece ninguno.

En cuanto al componente *vicc*, el problema, como seha mencionado, es el uso de python 2 y librerías obsoletas como Crypto en vez de PyCryptodome.

Respecto a las herramientas de simulación de tarjetas de la wiki de OpenSC[57], siguiendo los pasos y obviando el problema por la parte del driver, el error se encontraba en el paso de correr *jCardSim* con cualquiera de las alternativas (paso 8 de *IsoApplet* po ejemplo) y resultando en errores al no encontrar el módulo de la alternativa seleccionada.

Sin embargo, aunque se hubieran resuelto los problemas por la parte de la tarjeta simulada, el driver es de 2014, y en la página de su proyecto[58] se dice explícitamente que *"You can download the zip archive with the compiled binaries for Windows 7, 32 and 64 bits and Windows XP. In the zip file, you can find the DLL and the inf file, but you'll need devcon.exe for your architecture (it is not redistributable, but is part of WDK). The driver was ONLY tested under Win7 x64."* Aunque parece que hay gente que lo ha podido instalar en Windows 10, se dejó de probar las soluciones que empleaban este driver.

Entrando en el desarrollo de la prueba de concepto, el primer problema surgió al **compilar la extensión**, teniendo que modificar el archivo *scripts/build-utils.mjs* y sustituyendo la línea *const child = spawn('npx', args)* por lo siguiente

```
const child = command === "npx"? spawn( command: /^win/.test(process.platform) ? 'npx.cmd' : 'npx', args)
  : spawn(command, args);
```

Figura 40: Línea modificada en *scripts/build-utils.mjs*

Parece que se trata de un error al comprobar la plataforma en la que se compila.

[56] https://www.mankier.com/1/opensc-tool
[57] https://github.com/OpenSC/OpenSC/wiki/Smart-Card-Simulation
[58] https://www.codeproject.com/articles/134010/an-umdf-driver-for-a-virtual-smart-card-reader

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA